ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

TIENDA VIRTUAL PARA UNA GALERÍA DE ARTE

DEPARTAMENTO: Lenguajes y Sistemas Informáticos

DIRECTOR DEL PROYECTO: Juan Manuel Dodero Beardo

AUTORA DEL PROYECTO: Silvia María Garbarino de la Rosa

San Fernando, Abril 2013

Fdo: Silvia María Garbarino de la Rosa

Después de tanto tiempo
quiero dar las gracias:

A ***mis padres*** *y* ***tías***
por estar ahí siempre
que les he necesitado.

A ***mi marido***
por su comprensión,
por su paciencia y
por estar siempre conmigo.

A ***mi hijo***
por su alegría que cada
día comparte conmigo.

A ***mi tutor***
por darme la oportunidad
de terminar.

Gracias a todos de corazón.

Índice general

Listado de Tablas

Índice de figuras

Listado de Códigos

Parte I

Prolegómeno

Esta primera parte de la memoria de mi PFC va a contener dos capítulos:

- *El primero será una* **introducción** *en el que describiré en que consiste mi proyecto junto con los objetivos que me he marcado y por último, indicaré como he estructurado la presente memoria.*
- *El segundo capítulo estará dedicado a la* **planificación del proyecto** *incluyéndose la metodología que voy a emplear.*

Capítulo 1

Introducción

En este primer capítulo voy a realizar la presentación de mi *Proyecto de Fin de Carrera*. Explico cuáles son los motivos, objetivos y propósito del mismo, el producto final que deseo obtener y, por último, expongo cómo está estructurada la presente memoria.

1.1. Motivación

La realización de este proyecto supondrá la culminación de mis estudios de *Ingeniería Técnica en Informática de Gestión* y un fortalecimiento personal de los conocimientos adquiridos a lo largo de la carrera, tanto de planificación y análisis, como de programación y base de datos; así como la implicación en el desarrollo de un proyecto completo. Por otra parte, espero adquirir mayores conocimientos en el desarrollo de aplicaciones web, en comercio electrónico y en lenguajes de programación.

1.2. Propósito, objetivos y alcance del proyecto

En mi proyecto voy a desarrollar una aplicación de comercio electrónico (e-commerce) consistente en una Tienda Virtual. Esta podrá ser adaptada a distintos requisitos, permitiéndonos obtener una tienda para una *Galería de Arte* o para un *Artista*. De esta manera podré evaluar las ventajas de construir la aplicación respecto a inicializarla desde cero, usando otros frameworks o sin framework.

Con la tienda online se pretende conseguir abrir un nuevo mercado en Internet, para vender las obras de arte de la *Galería* o del *Artista* a través de la red, con la oportunidad de captar nuevos clientes y con el posible incremento de ventas que esto podría aportar.

A estos, se les dará la posibilidad de poder visitar y consultar un catálogo de obras online actualizado continuamente y la oportunidad de poder realizar los pedidos de manera más cómoda, rápida y segura.

La finalidad del proyecto no es construir un producto explotable comercialmente, sino poder aplicar al máximo los conocimientos aprendidos durante la trayectoria de mi carrera y adquirir nuevos conocimientos en el desarrollo de este.

Los *principales objetivos* de mi proyecto son:

1. Aprender a **desarrollar**, **implementar** y **poner en marcha una aplicación Web** que pueda tener una utilidad comercial, empleando la herramienta de desarrollo **Ruby on Rails** y los conocimientos adquiridos durante la carrera.

2. **Adaptar la aplicación a distintos requisitos** permitiéndome obtener otra a un menor coste.

3. Realizar un **análisis comparativo del framework *Ruby on Rails*** frente a otras alternativas, evaluando detalladamente las ventajas y características de este respecto a otros.

Para cumplir con ellos, hay que salvar una serie de objetivos o subtareas que componen el primer objetivo principal. Éstos se listan a continuación:

OBJ-0001	Gestión de artistas
Descripción	El sistema deberá gestionar los artistas de la galería.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.1: OBJ-0001 Gestión de artistas

OBJ-0002	Gestión de colecciones
Descripción	El sistema deberá gestionar las colecciones de las obras de arte.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.2: OBJ-0002 Gestión de colecciones

OBJ-0003	Gestión de obras
Descripción	El sistema deberá gestionar las obras de arte y las existencias de cada una de ellas.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.3: OBJ-0003 Gestión de obras

OBJ-0004	Gestión de secciones
Descripción	El sistema deberá gestionar las secciones a las que pertenecen las obras de arte.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.4: OBJ-0004 Gestión de secciones

OBJ-0005	Gestión de técnicas
Descripción	El sistema deberá gestionar las técnicas con las que se desarrollan las obras de arte.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.5: OBJ-0005 Gestión de técnicas

OBJ-0006	Gestión del catálogo
Descripción	El sistema deberá gestionar el catálogo de las obras de arte de la galería.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.6: OBJ-0006 Gestión del catálogo

OBJ-0007	Gestión de la cesta de la compra
Descripción	El sistema deberá gestionar el carrito de los clientes con las obras de arte que vayan añadiendo para posteriormente realizar sus compras.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.7: OBJ-0007 Gestión de la cesta de la compra

OBJ-0008	Gestión de etiquetas
Descripción	El sistema deberá gestionar las etiquetas con las que se clasificarán de las obras de arte.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.8: OBJ-0008 Gestión de etiquetas

OBJ-0009	Gestión de seguridad
Descripción	El sistema deberá gestionar la seguridad de la tienda virtual.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.9: OBJ-0009 Gestión de seguridad

OBJ-0010	Gestión de facturación
Descripción	El sistema deberá gestionar la facturación de los pedidos de los clientes.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.10: OBJ-0010 Gestión de facturación

OBJ-0011	Gestión de internacionalización
Descripción	El sistema deberá gestionar la internacionalización de la tienda virtual.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.11: OBJ-0011 Gestión de internacionalización

OBJ-0012	Gestión de usuarios
Descripción	El sistema deberá gestionar los usuarios de la tienda online.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.12: OBJ-0012 Gestión de usuarios

OBJ-0013	Gestión de quienes somos
Descripción	El sistema deberá gestionar el apartado de quienes somos.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.13: OBJ-0013 Gestión de quienes somos

OBJ-0014	Gestión de condiciones
Descripción	El sistema deberá gestionar las condiciones generales de la tienda online.
Subobjetivo	-
Importancia	Importante
Estabilidad	Alta
Comentarios	-

Figura 1.14: OBJ-0014 Gestión de condiciones

1.3. Producto final

La aplicación consistirá en generar un sistema web formado por:

1. Un *Front Office*, con el que interactuarán los clientes, formado por una Tienda Online. Donde podrán acceder al catálogo de las Obras de Arte y realizar sus compras.
2. Un *Back Office*, espacio restringido que permitirá al administrador gestionar toda la información del sistema.

1.4. Organización del documento

A continuación describo los contenidos del presente documento, así como del software entregado en soporte informático.

1.4.1. Organización de la memoria

En este apartado especifico como he estructurado esta memoria y como he organizado la información en función de los distintos capítulos que la forman.

El documento esta dividido en tres partes y cada una de ella en diferentes capítulos, los cuales describo a continuación:

I **Prolegómeno**

Capítulo 1 - Introducción. Contiene una breve descripción de mi Proyecto de Fin de Carrera, la motivación para su desarrollo, los objetivos y el propósito del mismo, así como la estructuración de la presente memoria.

Capítulo 2 - Planificación. Describe todos los aspectos relativos a la planificación del proyecto: metodología de desarrollo, planificación, organización, costes, riesgos.

II **Desarrollo**

Capítulo 3 - Análisis de requisitos. Recoge la especificación de requisitos del proyecto, así como:

- Un *análisis* de la relevancia de los principales *lenguajes de programación* dentro de la industria del software, realizando una valoración global de los resultados obtenidos.

- Una *comparativa* de los posibles *frameworks* que utilizan el lenguaje con el que voy a desarrollar mi proyecto.
- Un *análisis* de algunos de los *frameworks* más utilizados en comercio electrónico B2C.

Capítulo 4 - Diseño del sistema. Alberga la arquitectura general del sistema y los aspectos más destacables durante las fases de diseño.

Capítulo 5 - Implementación del sistema. Describiré los detalles más significativos de la fase de implementación de mi aplicación: entorno tecnológico, estructura de la aplicación, implementación del modelo, controlador y vista ...

Capítulo 6 - Pruebas del sistema. Contiene la documentación de los diferentes tipos de prueba llevados a cabo durante el desarrollo de la aplicación.

III **Epílogo**

Capítulo 7 - Manual de usuario. Detalla las instrucciones de uso del sistema.

Capítulo 8 - Manual de instalación y explotación. Especifica las instrucciones de instalación y explotación del sistema.

Capítulo 9 - Conclusiones. Muestra las lecciones aprendidas tras el desarrollo del proyecto, así como unas sugerencias sobre líneas de trabajo futuro.

Capítulo 10 - Anexos. La memoria contiene los siguientes anexos:

Anexo (A) Glosario de acrónimos. Listado de acrónimos que nos podemos encontrar en el presente documento.

Anexo (B) Glosario de definiciones. Contiene un catálogo de los términos utilizados junto con sus definiciones.

Anexo (C) Gemas. En este anexo nos vamos a encontrar con las descripciones de las gemas empleadas en el desarrollo de la aplicación.

Anexo (D) Lenguaje Ruby. Habla sobre el lenguaje Ruby y sus características.

Anexo (E) Metodología ágil. Describe la metodología ágil y la metodología scrum.

Anexo (F) Pruebas en Rails. Este anexo define las pruebas realizadas en Rails.

Anexo (G) Rails. Trata el entorno Rails y sus características.

Anexo (H) Relación de Casos de Uso y Mockups. Muestra un listado indicando para cada Caso de Uso el mockup que lo implementa.

Anexo (I) Test de usabilidad. Test de usabilidad realizado a los usuarios.

Glosario de términos. Contiene una relación de términos junto con las páginas en las que aparecen.

Bibliografía. Ofrece las referencias utilizada para el desarrollo del proyecto.

1.4.2. Organización del software

A continuación describo el contenido del CD que acompaña a la presente memoria. Éste contendrá las siguientes carpetas:

aplicaciones: Contiene las dos aplicaciones realizadas para el proyecto.

- **galeriaArte**: Código de la aplicación para la Galería de Artes.
- **galeriaArtista**: Código de la aplicación del Artista.

changelog: Incluye el fichero *Changelog* conteniendo el trabajo realizado cada día sobre el proyecto.

estadisticas_svn: Incluye las estadísticas de las diferentes carpetas del sistema software a través de la aplicación StatsSVN (*carpeta estadisticas_sist_software*) y el análisis de estadísticas de código incluido dentro de Ruby on Rails (*carpeta estadisticas_codigo*).

mockup: Contiene los mockup realizados para el presente proyecto.

usabilidad: Incluye los resultados del test de usuabilidad realizado a los usuarios.

Capítulo 2

Planificación

En este capítulo se describen todos los aspectos relativos a la planificación del proyecto: modelo de negocio, metodología que voy a emplear, organización, planificación temporal, costes del desarrollo del proyecto y gestión de riesgos.

2.1. Modelo de negocio

La aplicación que voy a desarrollar pertenece al *Modelo de Negocio* ***B2C Business-to-Consumer***, estrategia que desarrollan las empresas para establecer relaciones comerciales directas con sus clientes a través de Internet.

2.2. Metodología

Para asegurar el éxito, durante el desarrollo de la aplicación, no es suficiente contar con notaciones de modelado y herramientas, hace falta un elemento importante: la ***metodología de desarrollo***, la cual nos provee de una dirección a seguir para la correcta aplicación de los demás elementos.

Ésta nos va a ayudar a ***estructurar, planear*** y ***controlar*** el proceso de desarrollo del sistema.

Seguiré una ***Metodología Ágil*** [Marcelo Hernán, Schenone] (ver *anexo E*), ya que lo que necesito es poder gestionar mi proyecto de una forma ágil, debido a que hoy en día lo que se necesita es poder incorporar cambios con rapidez y en cualquier fase del proyecto.

Esta metodología consiste en un marco de trabajo conceptual de la *Ingeniería de Software* que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto.

Concretamente, seguiré el método ***Scrum*** (ver *anexo E*), el cual es iterativo e incremental, en el que divido el desarrollo de mi aplicación en ciclos llamados ***sprints*** y, en cada uno de ellos, trabajo sobre una lista de requisitos priorizada y al final de dicho ciclo, obtengo como resultado un producto terminado y entregable.

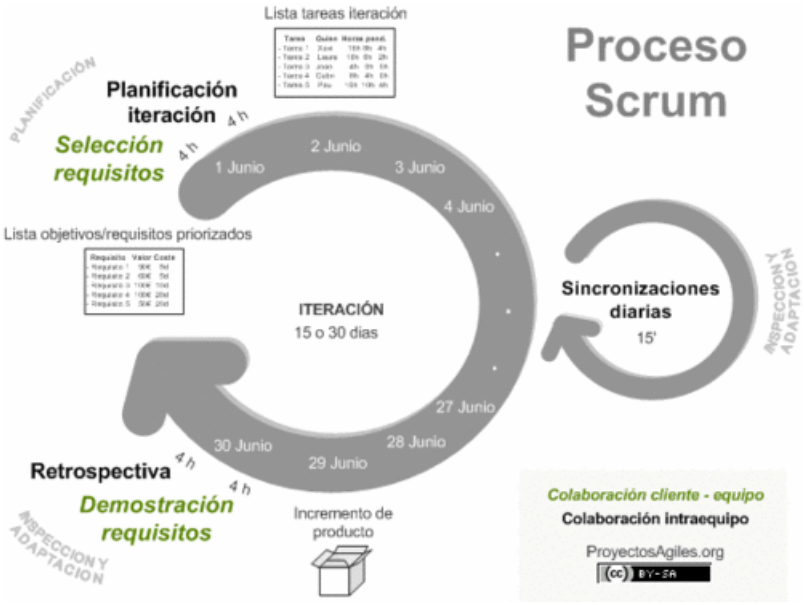


Figura 2.1: Diagrama del proceso Scrum

Cada iteración del ciclo de vida incluye:

- planificación,
- análisis de requerimientos,
- diseño,
- codificación,
- revisión y
- documentación.

2.3. Planificación del proyecto

En este apartado voy a establecer el orden de las tareas y realizaré una estimación del tiempo que creo necesario para la realización de mi proyecto. También mostraré un análisis de la desviación temporal, comparando los tiempos estimados con los que finalmente he empleado.

2.3.1. Objetivo inicial

El objetivo inicial de mi proyecto era desarrollar una aplicación de comercio electrónico (e-commerce) consistente en una *Tienda Virtual* para una *Galería de Arte*.

Finalmente, este objetivo se ha ampliado, desarrollando una aplicación que puede ser adaptada a distintos requisitos, permitiéndonos obtener una *Tienda Virtual* para una *Galería de Arte* o para un *Artista*.

2.3.2. División del trabajo

He diseñado un *diagrama* con la *Estructura de Desglose del Trabajo, WBS*, en el que muestro las actividades que he creído necesarias para completar mi proyecto.

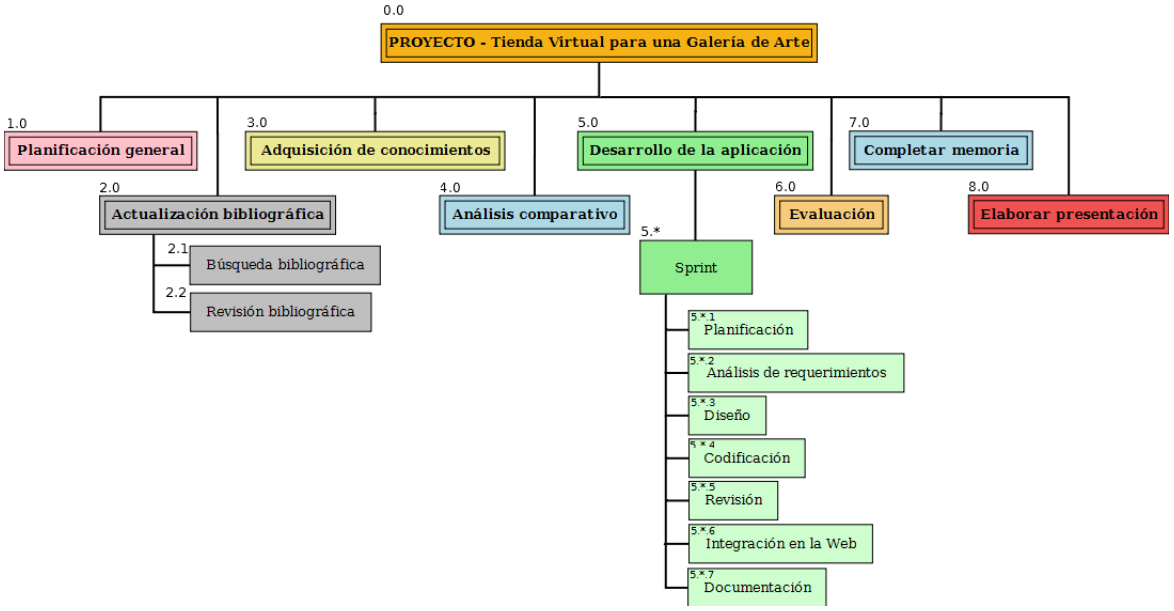


Figura 2.2: Estructura de desglose del trabajo

- **Planificación General.** Al comienzo del proyecto he realizado una planificación general para marcarme los pasos que tenía que seguir para que el proyecto fuese exitoso. Después, en cada *Sprint*, realizaré una planificación específica.

- **Búsqueda bibliográfica.** He realizado una búsqueda extensa de documentación a la cual poder acudir en cualquier fase del proyecto.

- **Adquisición de conocimientos.** Al comienzo del proyecto no tenía conocimientos acerca de *Comercio Electrónico*, *Ruby on Rails*, *CSS* ni de *LaTeX* [Burgos Pintos, Aris; Cornejo Barrios, Alicia; Gámez Mellado, Antonio; Otros]. Debido a ello he realizado un período inicial de aprendizaje y asimilación de conceptos básicos en

el que se incluyen, cursar la asignatura de *Comercio Electrónico de Ingeniería Informática* y participar en el *Taller de LaTeX* impartido por la Escuela durante la Semana de la Ingeniería.

- **Análisis comparativo.** Efectuaré un análisis comparativo del framework *Ruby on Rails* frente a otras alternativas, para poder evaluar las ventajas y características de este respecto a otros.

- **Desarrollo de la aplicación.** Una vez adquiridos los conocimientos básicos iré desarrollando la aplicación a través de diversos *Sprints* y en cada uno de ellos efectuaré las siguientes actividades.

  - **Planificación.** Para cada uno de los *Sprint* efectuaré una planificación específica. Esta la dividiré en dos pasos:
    - *Seleccionar los requisitos* a través de entrevistas con el cliente[1].
    - *Planificar el Sprint* elaborando una lista con las tareas que tenga que ejecutar para poder desarrollar los requisitos anteriormente marcados y asignándole, a cada una de las tareas, los tiempos de desarrollo con un margen de error para no verme demasiado comprometida.
  - **Análisis y diseño.** Estas actividades las iré realizando de forma paralela a la segunda y tercera de las fases anteriores, ya que conforme vaya analizando las necesidades del proyecto tendré que recurrir a la documentación recopilada para analizar su viabilidad, y para saber cómo poder llevarlas a cabo.
  - **Implementación.** En esta fase se realizará también una serie de pruebas para ir comprobando que lo que se vaya a implementar funcionaba correctamente. A la vez que las pruebas vayan funcionando, implementaré el código de la aplicación. Durante esta actividad sera necesario volver a las fases anteriores: para consultar cómo hacer las cosas o bien para dar solución a los problemas con los que me vaya encontrando.
  - **Revisión.** Durante las actividades anteriores y una vez terminadas éstas, realizaré revisiones del trabajo desarrollado con el fin de comprobar que se vayan cumplido los objetivos y los requisitos marcados.
  - **Integración en la Web.** El despliegue de la aplicación en la web lo realizaré en la plataforma ***Heroku***. Éste será continuo, ya que cada vez que vaya terminando un *Sprint* subiré al servidor el desarrollo realizado de la aplicación, comprobando también de esta manera su funcionamiento y pudiendo así el tutor o el cliente tener acceso a ella de forma rápida y sencilla en cualquier momento para su revisión.
    El despliegue de la *Galería de Arte* se puede visualizar en la siguiente dirección: `http://gadeirart.heroku.com/`.
  - **Documentación.** Durante todo el proceso del proyecto y al final de cada *Sprint* iré completando la documentación.

[1] *Cliente:* Galerista o Artista

- **Evaluación.** Al finalizar el proyecto elaboraré una evaluación del trabajo desarrollado, realizando una serie de análisis y comentando las posibles mejoras.
- **Completar memoria.** Una vez finalizado el proyecto repasaré y terminaré de completar la presente memoria.
- **Elaborar presentación.** Para dar por concluido el presente proyecto y poder realizar su presentación ante el tribunal, elaboraré una presentación del mismo.

A parte de las actividades anteriores mencionadas, realizaré:

- **Reuniones con el tutor.** Se irá realizando un seguimiento del proyecto. Éste se llevará a cabo mediante reuniones presenciales para ir viendo el avance y determinar los cambios que se pueden realizar en la aplicación. También haré uso del correo electrónico para comunicarme con el tutor, para que me pueda solventar las dudas que me vayan surgiendo.
- **Reuniones con el cliente.** Se realizarán reuniones con el cliente a lo largo del proyecto donde se definirán las *Historias de Usuario* y se validará el sistema.

2.3.3. Estimación de tiempos

Para la estimación temporal he tenido en cuenta los siguientes parámetros:

- *Días a la semana:* 5
- *Horas que puedo invertir por día:* 4h.

Todos los tiempos estarán expuestos a posibles imprevistos.

Actividad	**Duración estimada**
Planificación General y propuesta	4 días
Actualización bibliográfica	15 días
Adquisición previa de conocimientos	27 días
Análisis comparativo	3 días
Desarrollo de la aplicación	279 días
Integración en la web	15 días
Evaluación	2 días
Completar memoria	5 días
Elaborar Presentación	10 días
Esfuerzo total	360 días (72 semanas)

Tabla 2.1: Estimación de tiempos del proyecto

2.3.4. Identificación de Sprints

Para el correcto *desarrollo, control* y *seguimiento* de mi proyecto, he establecido una serie de *Sprints*[2] con los que podré ir viendo el avance y si se van cumpliendo los objetivos.

- *Sprint 0.* Contendrá las siguientes actividades:
  - Planificación general.
  - Elaboración de la propuesta de proyecto.
  - Bibliografía
  - Adquisición de conocimientos ...
- *Sprint 1 - Comienzo de la aplicación.* Algunas de las tareas de este *sprint* son:
  - Instalación de programas.
  - Creación de las BBDD.
  - Creación de la estructura de la aplicación.
  - CSS ...
- *Sprint 2 - Artistas.* Algunas tareas son (similares en los siguientes sprints):
  - Implementación Historias de Usuario.
  - Pruebas unitarias.
  - Pruebas funcionales.
  - Pruebas de integración ...
- *Sprint 3 - Colecciones.*
- *Sprint 4 - Obras.*
- *Sprint 5 - Secciones.*
- *Sprint 6 - Técnicas.*
- *Sprint 7 - Catálogo.*
- *Sprint 9 - Cesta de la compra.*
- *Sprint 10 - Etiquetado.*
- *Sprint 11 - Seguridad.*
- *Sprint 12 - Facturación.*

[2] *Sprints:* Hitos.

- *Sprint 13 - Internacionalización.*
- *Sprint 14 - Inicio.*
- *Sprint 15 - Contactar.*
- *Sprint 16 - Quienes somos.*
- *Sprint 17 - Condiciones.*
- *Sprint 18 - Panel Administración.*

2.3.5. Planificación temporal

La planificación temporal del proyecto la voy a mostrar mediante el *Diagrama de Gantt* que he preparado con la herramienta *OpenProj*. A través de éste, he establecido la asignación temporal prevista para la realización de las actividades, indicando los tiempos y aquellas tareas que pueden realizarse a lo largo del proyecto.

El desarrollo del *Diagrama de Gantt* de éste proyecto está representado en función de una persona, por lo que no se darán las relaciones de actividades solapadas ya que no será posible realizar simultáneamente dos tareas al mismo tiempo.

A continuación muestro la relación de actividades representadas en el *Diagrama de Gantt* junto con su duración, fecha de inicio y de terminación.

	Nombre	Duración	Inicio	Terminado
1	Planificación general	2 days	5/05/11 8:00	6/05/11 17:00
2	Propuesta proyecto	2 days	9/05/11 8:00	10/05/11 17:00
3	**⊟Bibliografía**	**15 days**	**11/05/11 8:00**	**31/05/11 17:00**
4	Búsqueda bibliográfica	10 days	11/05/11 8:00	24/05/11 17:00
5	Revisión bibliográfica	5 days	25/05/11 8:00	31/05/11 17:00
6	**⊟Adquisición de conocimientos**	**27 days**	**1/06/11 8:00**	**7/07/11 17:00**
7	Comercio electrónico	2 days	1/06/11 8:00	2/06/11 17:00
8	Ruby on Rails	15 days	3/06/11 8:00	23/06/11 17:00
9	Latex	5 days	24/06/11 8:00	30/06/11 17:00
10	CSS	5 days	1/07/11 8:00	7/07/11 17:00
11	Análisis comparativo	3 days	8/07/11 8:00	12/07/11 17:00
12	**⊞Desarrollo aplicación**	**279 days**	**13/07/11 8:00**	**27/09/12 17:00**
35	Integración en la Web	15 days	28/09/12 8:00	18/10/12 17:00
36	Evaluación	2 days	19/10/12 8:00	22/10/12 17:00
37	Completar memoria	5 days	23/10/12 8:00	29/10/12 17:00
38	Presentación proyecto	10 days	30/10/12 8:00	13/11/12 17:00

Figura 2.3: Relación actividades Diagrama de Gantt

En la figura 2.3, para que quede un poco más claro, podemos observar la relación de actividades del proyecto en general.

Y en la figura 2.4 les muestro, en detalle, la relación de actividades del diagrama para la actividad *Desarrollo de la aplicación* que es la que más tiempo me va a llevar.

	Nombre	Duración	Inicio	Terminado
12	**Desarrollo aplicación**	279 days	**13/07/11 8:00**	**27/09/12 17:00**
13	**Sprint 1 - Comienzo**	6 days	**13/07/11 8:00**	**20/07/11 17:00**
14	Instalación programas	2 days	13/07/11 8:00	14/07/11 17:00
15	Creación BBDD	1 day	15/07/11 8:00	15/07/11 17:00
16	Layout	1 day	18/07/11 8:00	18/07/11 17:00
17	CSS	2 days	19/07/11 8:00	20/07/11 17:00
18	Sprint 2 - Artistas	10 days	21/07/11 8:00	3/08/11 17:00
19	Sprint 3 - Colecciones	5 days	4/08/11 8:00	10/08/11 17:00
20	Sprint 4 - Obras	20 days	11/08/11 8:00	7/09/11 17:00
21	Sprint 5 - Secciones	5 days	8/09/11 8:00	14/09/11 17:00
22	Sprint 6 - Técnicas	5 days	15/09/11 8:00	21/09/11 17:00
23	Sprint 7 - Catálogo	30 days	22/09/11 8:00	3/11/11 17:00
24	Sprint 9 - Cesta de la compra	10 days	4/11/11 8:00	17/11/11 17:00
25	Sprint 10 - Etiquetado	7 days	18/11/11 8:00	28/11/11 17:00
26	Sprint 11 - Autenticación	20 days	29/11/11 8:00	2/01/12 17:00
27	Sprint 12 - Compra y procesamiento de pedidos	30 days	3/01/12 8:00	15/02/12 17:00
28	Sprint 13 - Internacionalización	20 days	16/02/12 8:00	15/03/12 17:00
29	Sprint 14 - Memoria	100 days	16/03/12 8:00	12/09/12 17:00
30	Sprint 15 - Menú Inicio	2 days	13/09/12 8:00	14/09/12 17:00
31	Sprint 16 - Menú Contactar	2 days	17/09/12 8:00	18/09/12 17:00
32	Sprint 17 - Menú Quienes Somos	2 days	19/09/12 8:00	20/09/12 17:00
33	Sprint 18 - Menú Condiciones Generales	2 days	21/09/12 8:00	24/09/12 17:00
34	Sprint 19 - Menú Administración	3 days	25/09/12 8:00	27/09/12 17:00

Figura 2.4: Relación actividades Diagrama de Gantt - Desarrollo de la aplicación

Seguidamente se muestra el *Diagrama de Gantt del proyecto*, el cual, para que quede un poco más claro, lo he dividido en tres gráficos.

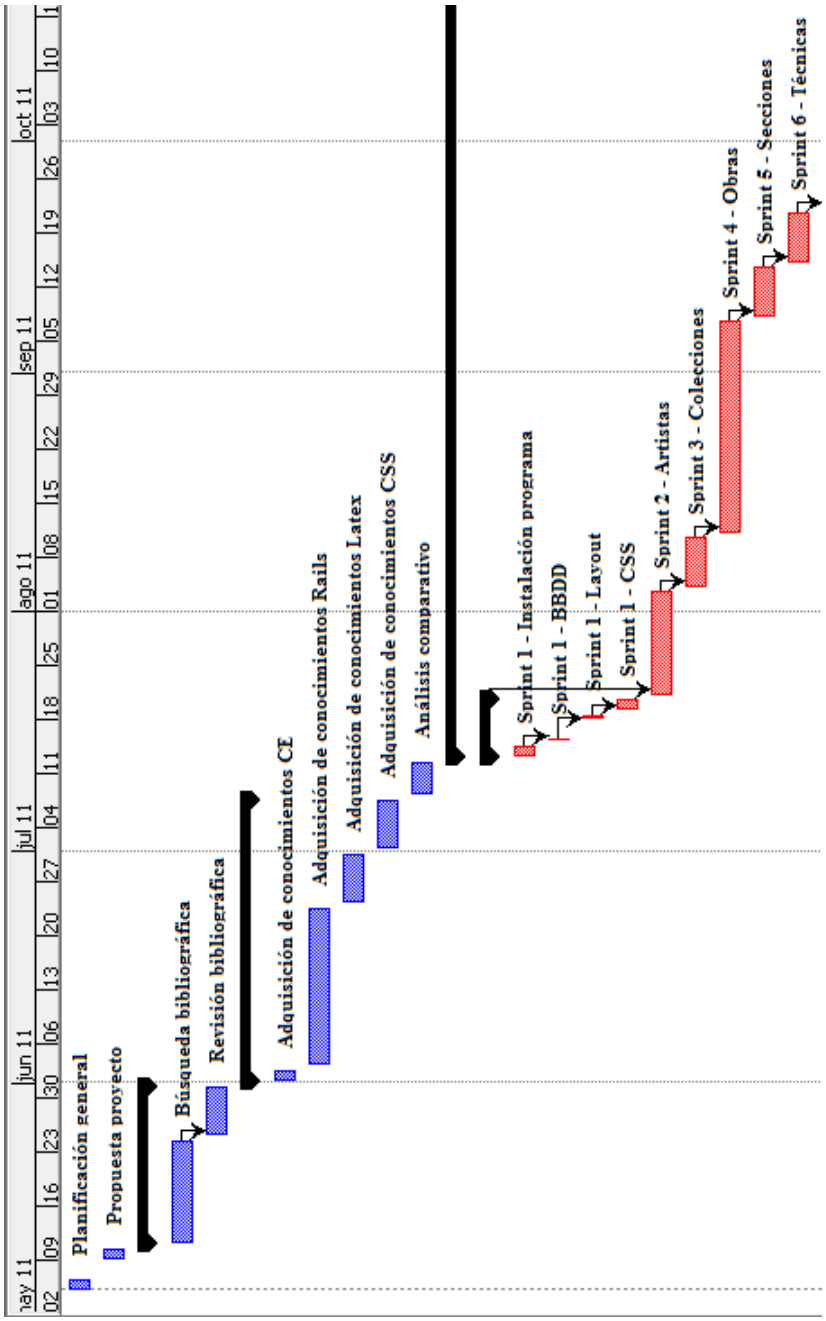


Figura 2.5: Diagrama de Gantt

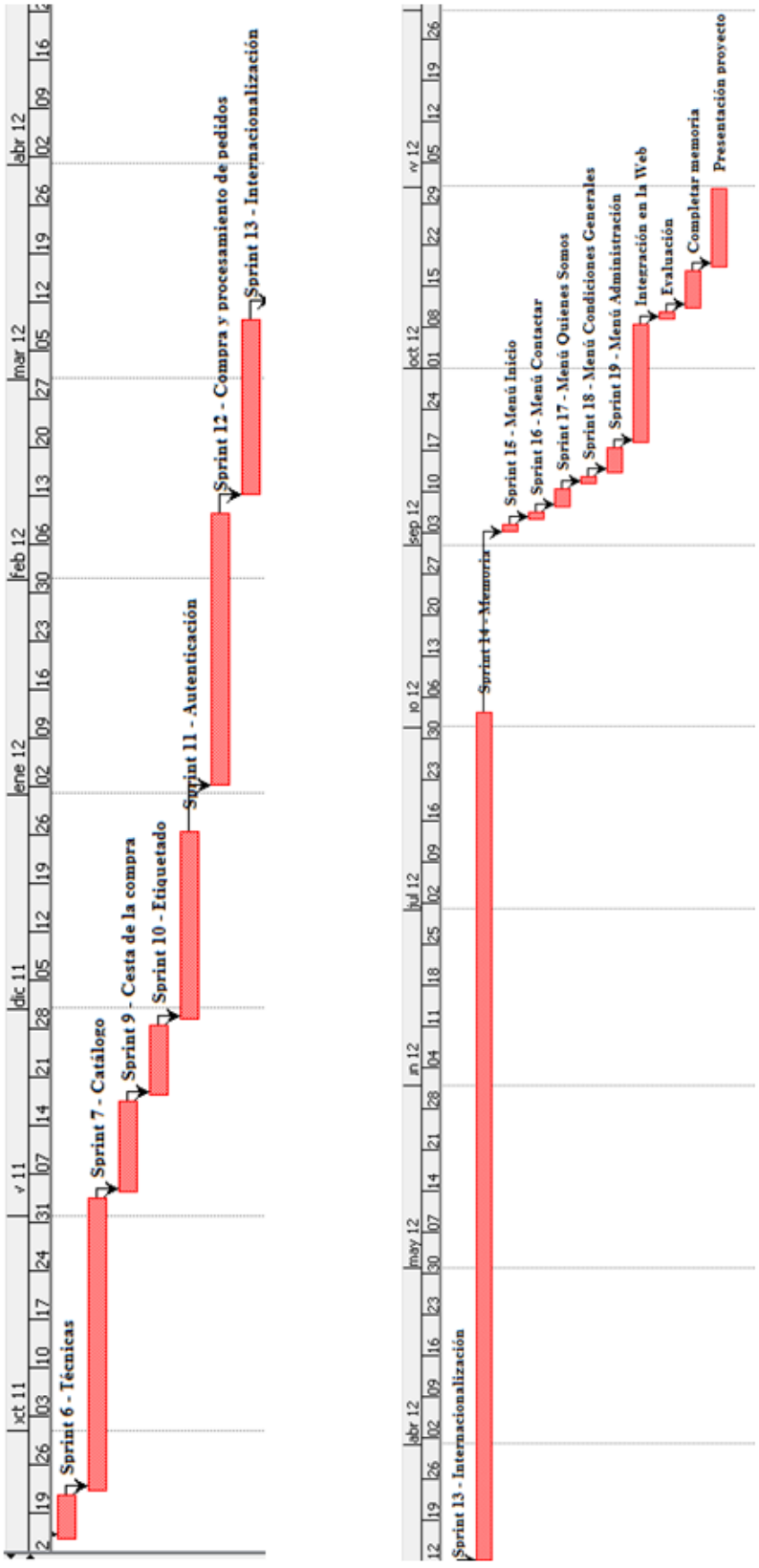


Figura 2.6: Diagrama de Gantt

Podemos observar en el gráfico, en color azul, las tareas que pueden realizarse o desplazarse a lo largo del proyecto sin que ello suponga un retraso en la realización del mismo, y en color rojo las tareas críticas, las cuales deben realizarse en el orden y la posición indicadas.

2.3.6. Análisis de la desviación temporal

A continuación muestro una comparación de los tiempos estimados con los finalmente empleados.

Sprint	**Días estimados**	**Horas estimadas** (4h/día)	**Horas invertidas** *Hor*	*Min*	**Diferencia horas** *Hor*	*Min*
Planificación general	2	8	5h	30m	2h	30m
Propuesta proyecto	2	8	9h		-1h	
Actualización bibliográfica	15	60	25h		35h	
Adquisición de conocimientos	27	108	-		108h	
Análisis comparativo	3	12	-		12h	
Desarrollo de la aplicación	279	1116	1028h	30m	87h	30m
* Sprint 1 - Comienzo	6	24	15h		9h	
* Sprint 2 - Artistas	10	40	30h	15m	9h	45m
* Sprint 3 - Colecciones	5	20	15h	45m	4h	15m
* Sprint 4 - Obras	20	80	68h	30m	11h	30m
* Sprint 5 - Secciones	5	20	12h	15m	7h	45m
* Sprint 6 - Técnicas	5	20	18h		2h	
* Sprint 7 - Catálogo	30	120	123h	30m	-3h	30m
* Sprint 9 - Cesta de la compra	10	40	45h	30m	-5h	30m
* Sprint 10 - Etiquetado	7	28	26h		2h	
* Sprint 11 - Autenticación	20	80	71h	30m	8h	30m
* Sprint 12 - Compra y ...	30	120	130h		-10h	
* Sprint 13 - Internacionalización	20	80	85h	15m	-5h	15m
* Sprint 14 - Memoria	100	400	351h	15m	48h	45m
* Sprint 15 - Menú Inicio	2	8	6h		2h	
* Sprint 16 - Menú Contactar	2	8	3h		5h	
* Sprint 17 - Menú Quienes Somos	2	8	7h	30m		30m
* Sprint 18 - Menú Condiciones	2	8	6h	15m	1h	45m
* Sprint 19 - Administración	3	12	13h		-1h	
Integración en la web	15	60	93h		-33h	
Evaluación	2	8	-		8h	
Completar memoria	5	20	34		-14h	
Elaborar Presentación	10	40	-		40h	
Desviación temporal	**360**	**1440**	**1195h**		**245h**	

Tabla 2.2: Estudio del tiempo del proyecto

La *adquisición de conocimientos* no está contabilizada directamente en su apartado, ya que los he ido adquiriendo con anterioridad al comienzo del proyecto: cursando la asignatura de *Comercio Electrónico de Ingeniería Informática* y realizando el *Taller de LaTeX* impartido por la Escuela, y durante la realización del proyecto, contabilizando este tiempo en los sprints correspondientes.

Las horas estimadas e invertidas en el *análisis comparativo* y la *evaluación* están contabilizadas dentro del *Sprint 14 - Memoria*.

En resumen, puedo comentar que he estimado más o menos el número de semanas necesarias para la realización del proyecto y de la memoria. Teniendo en cuenta, que en este estudio están contabilizadas las 10 semanas que invertiré próximamente para la realización de la presentación del proyecto.

Semanas estimadas	72 semanas
Semanas invertidas	60 semanas
Desviación temporal	12 semanas

Tabla 2.3: Estudio de la desviación temporal del proyecto

2.4. Organización del proyecto

En esta sesión hablaré sobre la relación de personas (roles) involucradas en el proyecto así como de los recursos inventariables utilizados.

2.4.1. Participantes

2.4.1.1. Estructura interna

En la figura 2.7 se muestra la jerarquía del equipo, así como la relación existente entre ellos.

2.4.1.2. Roles y responsabilidades

Podemos distinguir los siguientes roles en función de su perfil, según lo recogido en *Métrica v.3 - Participantes [PAE. Portal Administración Electrónica (2011)]*.

Supervisor del proyecto. Se ocupará de guiar y supervisar el proyecto. Este rol lo desempeñará el tutor del proyecto J.M. Dodero.

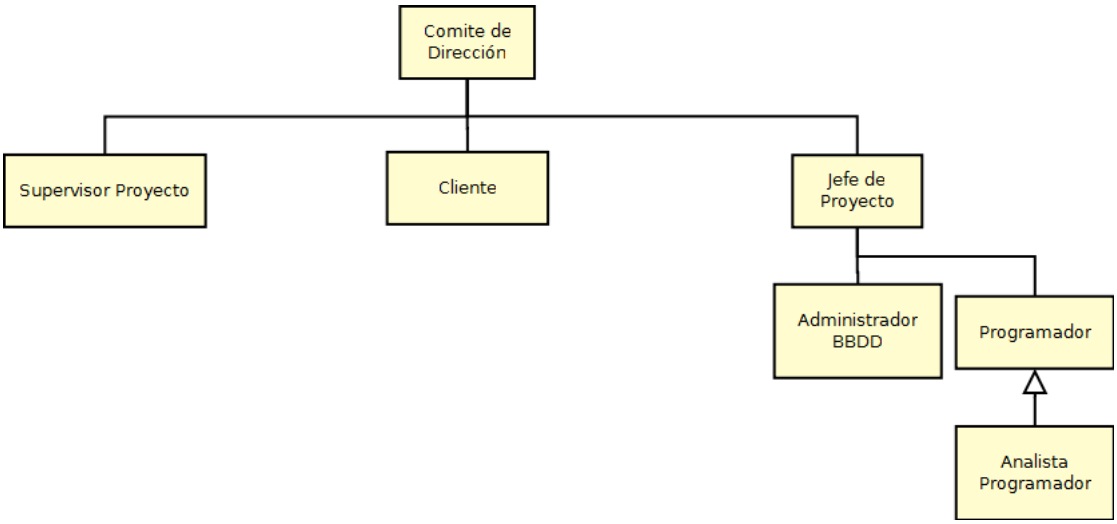


Figura 2.7: Organigrama de la Organización

Cliente. Aporta información sobre las necesidades planteadas y valida los resultados con el fin de garantizar la identificación, comprensión e incorporación de todos los requisitos con las prioridades adecuadas.

Para el proyecto el cliente es un artista-galerista.

Cada uno de los roles que voy a describir a continuación serán desempeñados por mi persona en el desarrollo del proyecto.

Perfil Jefe de Proyecto. Ejerce labores de coordinación y dirección de equipos humanos especializados en la realización de actividades propias de un proceso o interfaz de PAE. Portal Administración Electrónica (2011). La figura principal es el Jefe de Proyecto, el cual recibe el apoyo de los distintos responsables durante la realización de procesos o determinadas actividades a lo largo del proyecto.

Este perfil, vendrá representado por mi persona, con las siguientes responsabilidades:

- Realizar la estimación del esfuerzo necesario para llevar a cabo el proyecto.
- Seleccionar la estrategia de desarrollo, determina la estructura del mismo seleccionando los procesos principales de PAE. Portal Administración Electrónica (2011) que lo integran.
- Fijar el calendario de hitos y entregas y establece la planificación del proyecto.
- Es el encargado de dirigir el proyecto, realizando las labores de seguimiento y control del mismo, revisión y evaluación de resultados y coordinación del equipo de proyecto.
- Se ocupa también de la gestión y resolución de incidencias que puedan surgir durante el desarrollo del proyecto así como de la actualización de la planificación inicial.

Perfil Analista. La responsabilidad del Analista es elaborar un catálogo detallado de requisitos que permita describir con precisión el sistema de información, para lo cual mantendrán entrevistas y sesiones de trabajo con los clientes[3]. Estos requisitos permiten al analista elaborar los distintos modelos que sirven de base para el diseño, obteniendo los modelos de clases e interacción de objetos en análisis orientado a objeto. Así mismo, realizan la especificación de las interfaces entre el sistema y el usuario.

Dentro de este perfil, se distinguirán los siguientes roles:

- **Analista**
- **Administrador de Bases de Datos.** Participa en la obtención del diseño físico de datos, definiendo la estructura física de datos que utilizará el sistema a partir del modelo lógico de datos normalizado o del modelo de clases, teniendo presentes las características específicas del sistema de gestión de base de datos concreto a utilizar, los requisitos establecidos para el sistema de información, y las particularidades del entorno tecnológico, se consiga una mayor eficiencia en el tratamiento de los datos.

  Si se va a realizar una migración de datos colabora con el equipo de proyecto estimando los volúmenes de las estructuras de datos implicadas, definiendo los mecanismos de migración y carga inicial de datos y participando activamente en su realización.

  Una vez que el sistema está en producción se ocupa de la gestión y operativa asociada a las bases de datos y al software en el que están implementadas.

Perfil Programador. La función del programador es construir el código que dará lugar al producto resultante en base al diseño técnico realizado por el analista o analista programador, generando también el código asociado a los procedimientos de migración y carga inicial de datos. Igualmente se encarga de la realización de las pruebas unitarias y participa en las pruebas de conjunto de la aplicación.

2.4.2. Equipo informático

2.4.2.1. Hardware

El hardware del equipo informático utilizado para la realización del proyecto presenta las siguientes características:

- Ordenador portátil Aspire 5740G, Intel Core i3 processor 330M, 4GB RAM, 320 GB HDD.
- Impresora Hewlett Packard Deskjet 5550.

[3] *Cliente:* Galerista o Artista

2.4.2.2. Software

El software empleado para el desarrollo bajo el entorno Ubuntu es el siguiente:

- *Lenguajes*: Ruby 1.8.7, HTML, CSS, JavaScript (ver sección 5.1.1).
- *Framework*: Rails 3.0.9 (ver sección 3.6.2.1).
- *Base de datos*: MySql Server Versión 5.1.54 - 1ubuntu4 (ver sección 5.1.2).
- *Sistema operativo*: Ubuntu Linux 11.04.
- *Navegador*: Firefox 4.0.

La justificación de la elección de cada uno de ellos podemos verla en la secciones que aparecen entre paréntesis o a continuación:

- **Sistema Operativo:** Un elemento básico e imprescindible a la hora de construir una aplicación es la elección del S.O.

  Para el desarrollo de mi aplicación he seleccionado el S.O. Ubuntu y algunas de las razones son:

  - Instalación de programas sencilla.
  - No necesita antivirus.
  - Es potente y fiable.
  - Coste cero.
  - Integración con el resto de componentes: lenguajes de programación, bases de datos y herramientas de desarrollo.

- **Navegador:** La aplicación web se construirá bajo el requisito de independencia del navegador. Es por ello por lo que se utilizarán varios navegadores durante el desarrollo probando así que se cumple dicho requisito.

2.5. Costes

En esta sección voy a elaborar una estimación de los costes del proyecto.

Para ello evaluaré cada uno de los recursos implicados en el desarrollo: recursos humanos, herramientas, costes indirectos propios de las empresas que trabajan en este sector (conexión a internet, material de oficina, recambios de impresora...) y los costes del equipo informático (ordenador e impresora), ya que aunque las empresas dedicadas al desarrollo de software cuentan con el equipo informático necesario, hay que tenerlos en

cuenta porque puede surgir cualquier imprevisto, durante el desarrollo del proyecto, que implique el tener que realizar una nueva adquisición.

Para obtener una estimación de los costes de personal he consultado diferentes web dedicadas a ofertas de trabajo (infojobs, tecnoempleo...) y he obtenido que la media salarial ronda los 25.000€/brutos/año, a lo que hay que añadir el coste de la Seguridad Social que se considera un tercio de la base de cotización. Desglosando se obtiene:

- *Salario mensual.* Anualmente se ha estimado un sueldo de 25.000€, que repartidos en 14 pagas hacen que el salario mensual sea de 1.785,72€.
- *Seguridad social.* El 33 % del sueldo mensual equivale a 589.29€/mes.

Luego, el coste total mensual del personal asciende a 2.375€.

Para los costes del equipo informático he tenido en cuenta el tiempo en el que se amortiza. Un equipo suele tener una vida de 3 años, por lo que si su coste es de 1.500€ se amortizarán anualmente 500€.

Los costes indirectos van a suponer un 10 % del coste del personal y los costes de las herramientas necesarias para el proyecto, no los tengo en cuenta ya que la mayoría son software libre o la empresa ya cuenta con ellos.

Para poder determinar el tiempo empleado en el desarrollo de la aplicación, he ido llevando el control de las horas invertidas en *Assembla*, habiendo utilizado un total de 1.195h horas correspondiente a 30 semanas.

Para el cálculo he tenido en cuenta los siguientes parámetros:

- *Personas implicadas en el proyecto:* 1
- *Días a la semana:* 5
- *Horas invertidas por día:* 8h.
- *Meses empleados:* 7 meses (30 semanas).

A continuación, podemos ver la información recogida en la siguiente tabla:

Nombre	**Descripción**	**Tipo**	**Coste**	**Total**
Personal	Personal cualificado y especializado	Humano	2.375€/mes	16.625€
Equipo informático	PC e impresora	Activo	42€/mes	294€
Costes indirectos	Tinta, luz...	Material	10 % del coste del personal	1.662,5€
			Total	18.581,50€

Tabla 2.4: Costes del proyecto

2.6. Gestión de riesgos

Un proceso efectivo de gestión de riesgos es un importante elemento para que un proyecto de software sea exitoso.

La *gestión de riesgos* [Alarcos] nos permite definir de forma estructurada, operacional y organizada, una serie de actividades para gestionar los riesgos del proyecto a lo largo de su ciclo de vida.

El propósito es identificar los riesgos que se puedan presentar en el desarrollo del proyecto, analizarlos, calcular la exposición y, en base a ello, poder priorizarlos, para establecer estrategias de control y resolución que permitan ejercer una correcta supervisión de los mismos.

2.6.1. Evaluación de riesgos

2.6.1.1. Identificación de riesgos

Lo primero que voy a hacer es identificar cada uno de los posibles riesgos que pueden afectar al proyecto.

- *Riesgos en la planificación.* Que puede conllevar el fracaso del proyecto. Esto es, que no se cumplan los objetivos con las estimaciones de tiempo y recursos.

  Causas:

  - Planificación muy optimista.
  - La planificación no incluye tareas necesarias.
  - Decisiones de planificación impuestas por el cliente.
  - Infraestimación de tiempos de desarrollo de las tareas.
  - Un retraso en una tarea produce retrasos en cascada en las tareas dependientes.

- *Riesgos en la organización y gestión.*

  - La planificación es demasiado mala para ajustarse a la velocidad de desarrollo deseada.
  - El plan del proyecto se abandonan por la presión, llevando al caos y a un desarrollo ineficiente.

- *Riesgos en la infraestructura.* Causado por:

  - Los espacios están disponibles pero no son adecuados (por ejemplo, cableado de red).

  - Herramientas de desarrollo no disponibles o funcionamiento/prestaciones indeseadas.
  - La curva de aprendizaje para la nueva herramienta de desarrollo es más larga de lo esperado.

- *Riesgos en los clientes.* Causas:
  - Problemas con el personal de desarrollo: entregas fuera de plazo, cree que no entienden bien sus requisitos, dificultad en la comunicación, etc.
  - El ciclo de revisión/decisión del cliente es más lento que lo previsto.
  - No participación del cliente en la revisión, implicando requisitos inestables.
  - El cliente insiste en tomar decisiones técnicas que alargan la planificación.
  - El cliente intenta controlar el ritmo de desarrollo y producción.
  - El cliente insiste en nuevos requisitos.
  - El tiempo de comunicación del cliente (por ejemplo, tiempo para responder a las preguntas para aclarar requisitos) es más lento del esperado.
  - En el último momento, al cliente no le gusta el producto, por lo que hay que volver a diseñarlo o a construirlo.
  - No se ha solicitado información al cliente, por lo que el producto al final no se ajusta a las necesidades de éste, y hay que volver a crearlo.

- *Riesgos en los requisitos:* Fundamentalmente debidos a los cambios en los requisitos, derivados del no establecimiento de los requisitos del proyecto y cambiarlos a medida que éste avanza. Puede deberse a:
  - La no correcta definición de los requisitos y su redefinición conforme avanza el proyecto.
  - Se añaden requisitos extras.
  - Las partes del proyecto que no se habían especificado claramente consumen más tiempo de lo previsto.

- *Riesgos en el producto.* Causas:
  - Utilizar lo último en informática alarga la planificación de forma impredecible.
  - El desarrollo de funciones software erróneas requiere volver a diseñarlas y a implementarlas.
  - El desarrollo de una interfaz de usuario inadecuada requiere volver a diseñarla y a implementarla.
  - El desarrollo de funciones software innecesarias alarga la planificación.
  - El trabajo con un entorno software desconocido causa problemas no previstos.

- *Riesgos en el personal.*
  - Problemas con los clientes.
  - Las tareas preliminares no se acaban a tiempo (por ejemplo, formación).
  - Falta de especialización afecta a los defectos y la necesidad de repetir tareas.
  - El personal necesita tiempo extra para aprender nuevos lenguajes o herramientas.
  - Falta de motivación del personal del equipo de proyecto.
- *Riesgos en diseño e implementación.* Causados por:
  - No se considera necesario realizar el diseño y se pasa directamente a la implementación.
  - El diseño se realiza más por cumplir una metodología que porque se considere necesario o útil.
  - Se realiza un diseño simple y preliminar que no resuelve los problemas que obligan más tarde a rediseñar y volver a codificar.
  - El diseño es demasiado detallado y posee complicaciones innecesarias.
  - Un mal diseño implica volver a diseñar e implementar.
  - No es posible implementar la funcionalidad deseada en el lenguaje de programación.
  - No es posible integrar los componentes desarrollados y es necesario rediseñar y repetir algunas tareas de programación.
- *Riesgos en el proceso.* Causas:
  - La falta de un seguimiento exacto del progreso hace que se desconozca que el proyecto esté retrasado hasta que está muy avanzado.
  - La falta de rigor (ignorar los fundamentos y estándares del desarrollo de software) en los procesos provoca errores de comunicación, problemas de calidad... que conducen a la repetición de tareas.
  - El exceso de rigor hace que se progrese más lentamente de lo previsto.
  - La falta de entusiasmo en la gestión de los riesgos impide detectar los riesgos más importantes del proyecto.
  - La gestión de los riesgos del proyecto consume más tiempo del esperado.

2.6.1.2. Análisis de riesgos

Una vez identificados los riesgos en el apartado 2.6.1.1, el siguiente paso es analizar cada uno de ellos para determinar su impacto.

Para ello se realizarán los siguientes cálculos:

- **Cálculo del impacto o exposición a un riesgo.**

  Impacto del riesgo = Probabilidad del riesgo × Magnitud del riesgo (pérdida)

- **Estimación de la magnitud de la pérdida:** Efecto sobre los objetivos del proyecto en caso de ocurrir. Se computará en unidades de tiempo (retraso).

- **Estimación de la probabilidad de un riesgo:** Porcentaje de probabilidad de que ocurra un riesgo, en una escala de 1 a 100, siendo 1: baja probabilidad y 100: máxima probabilidad.

En las siguientes tablas se recogen los parámetros mencionados para cada riesgo:

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Planificación muy optimista	50 %	5 semanas	2,5 semanas
Planificación no incluye tareas necesarias	25 %	4 semanas	1 semana
Decisiones impuestas por cliente	25 %	3 semanas	0,75 semana
Infraestimación tiempos desarrollo tareas	50 %	4 semanas	2 semanas
Retrasos en cascada	10 %	6 semanas	0,6 semana

Tabla 2.5: Análisis de riesgos en la planificación

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Planificación mala	10 %	5 semanas	0,5 semana
Abandono plan proyecto	10 %	5 semanas	0,5 semana

Tabla 2.6: Análisis de riesgos en la organización

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Espacios no adecuados	5 %	5 semanas	0,25 semana
Herramientas no disponibles o funcionamiento/prestaciones indeseadas	50 %	5 semanas	2,5 semanas
Curva de aprendizaje más larga	10 %	6 semanas	0,6 semana

Tabla 2.7: Análisis de riesgos en la infraestructura

Riesgo	Probabilidad	Magnitud de la pérdida	Exposición al riesgo
No correcta definición de requisitos	50 %	5 semanas	2,5 semanas
Se añaden requisitos extra	50 %	15 semanas	7,5 semanas
Partes del proyecto no especificadas claramente	45 %	10 semanas	4,5 semanas

Tabla 2.8: Análisis de riesgos en los requisitos

Riesgo	Probabilidad	Magnitud de la pérdida	Exposición al riesgo
Falta seguimiento del progreso	20 %	10 semanas	2 semanas
Repetición de tareas	10 %	10 semanas	1 semana
Exceso de rigor	30 %	10 semanas	3 semanas
Incapacidad de detección riesgos más importantes	40 %	7 semanas	2,8 semanas
La gestión de los riesgos necesita más tiempo del esperado	40 %	4 semanas	1,6 semanas

Tabla 2.9: Análisis de riesgos en el proceso

Riesgo	Probabilidad	Magnitud de la pérdida	Exposición al riesgo
Problemas con el equipo de desarrollo	20 %	5 semanas	1 semana
Lento ciclo de revisión/decisión del cliente	30 %	3 semanas	0,9 semana
No participación del cliente en la revisión	10 %	4 semanas	0,4 semana
El cliente insiste en tomar decisiones técnicas que alargan la planificación	30 %	5 semanas	1,5 semanas
El cliente intenta controlar el ritmo de desarrollo y producción	5 %	5 semanas	0,25 semana
El cliente insiste en nuevos requisitos	25 %	5 semanas	1,25 semanas
Tiempo de comunicación del cliente lento	5 %	4 semanas	0,20 semana
Al cliente no le gusta el producto	5 %	5 semanas	0,25 semana
No se ha solicitado información al cliente	15 %	6 semanas	0,9 semana

Tabla 2.10: Análisis de riesgos en los clientes

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Problemas con los clientes	20 %	5 semanas	1 semana
Las tareas preliminares no se acaban a tiempo	30 %	5 semanas	1,5 semanas
Falta de especialización	40 %	4 semanas	1,6 semanas
El personal necesita tiempo extra de aprendizaje	50 %	4 semanas	2 semanas
Falta de motivación	20 %	5 semanas	1 semana

Tabla 2.11: Análisis de riesgos en el personal

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Utilizar lo último en informática	5 %	5 semanas	0,25 semana
Desarrollo funciones erróneas	25 %	10 semanas	2,5 semanas
Desarrollo interfaz inadecuada	10 %	5 semanas	0,5 semana
Desarrollo funciones innecesarias	15 %	10 semanas	1,5 semanas
Trabajo en entorno desconocido	10 %	5 semanas	0,5 semana

Tabla 2.12: Análisis de riesgos en el producto

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Pasar directamente a implementar	15 %	10 semanas	1,5 semanas
Diseño por cumplir	20 %	8 semanas	1,6 semanas
Diseño simple y preliminar	15 %	10 semanas	1,5 semanas
Diseño demasiado detallado y con complicaciones innecesarias	30 %	3 semanas	0,9 semana
Imposibilidad implementar funcionalidad	40 %	4 semanas	1,6 semanas
No es posible integrar los componentes desarrollados	40 %	5 semanas	2 semanas

Tabla 2.13: Análisis de riesgos en diseño e implementación

2.6.1.3. Priorización de riesgos

De acuerdo con el análisis recogido en el punto anterior y tras un estudio de qué causas pueden influir en otras, he establecido el orden de los riesgos de mayor a menor prioridad, que precisan su control de manera inminente:

1. Planificación muy optimista.
2. Herramientas de desarrollo no disponibles o funcionamiento/prestaciones indeseadas.
3. La no correcta definición de los requisitos y su redefinición conforme avanza el proyecto.
4. El personal necesita tiempo extra para aprender nuevas herramientas.
5. La falta de un seguimiento exacto del progreso hace que se desconozca que el proyecto esté retrasado hasta que está muy avanzado.
6. No es posible integrar los componentes desarrollados y es necesario rediseñar y repetir algunas tareas de programación.
7. No es posible implementar la funcionalidad deseada en el lenguaje de programación.
8. Se realiza un diseño simple y preliminar que no resuelve los problemas que obligan más tarde a rediseñar y volver a codificar.

A continuación muestro la tabla de estimación de riesgos priorizada.

Riesgo	**Probabilidad**	**Magnitud de la pérdida**	**Exposición al riesgo**
Planificación muy optimista	50 %	5 semanas	2,5 semanas
Herramientas no disponibles o funcionamiento/prestaciones indeseadas	50 %	5 semanas	2,5 semanas
No correcta definición de requisitos	50 %	5 semanas	2,5 semanas
El personal necesita tiempo extra de aprendizaje	50 %	4 semanas	2 semanas
Falta seguimiento del progreso	20 %	10 semanas	2 semanas
No es posible integrar los componentes desarrollados	40 %	5 semanas	2 semanas
Imposibilidad implementar funcionalidad	40 %	4 semanas	1,6 semanas
Diseño simple y preliminar	15 %	10 semanas	1,5 semanas

Tabla 2.14: Estimación de riesgos priorizados

2.6.2. Control de riesgos

En la sección 2.6.1.3 se definió un listado de los principales riesgos que se deben controlar. La presente sección tiene dos objetivos principales, por un lado *describir aquellas medidas que minimizarán la probabilidad de aparición de estos riesgos* y por otro, *describir aquellas acciones a llevar a cabo en el caso de que se produzca alguno de ellos.*

2.6.2.1. Reducción de riesgos

- **Riesgo 1: Planificación muy optimista.**
  A la hora de realizar la planificación será obligatorio asumir una actitud objetiva y prepararse para el caso en que las cosas no salgan tan bien como se espere, con el objetivo de realizar una planificación realista. Para todo ello, será indispensable respetar los tiempos de las tareas analizadas. La planificación deberá de ser supervisada durante su desarrollo y corregida una vez realizada.

- **Riesgo 2: Herramientas de desarrollo no disponibles o funcionamiento/-prestaciones indeseadas.**
  Para evitar problemas con las herramientas de desarrollo o con las prestaciones, el proyecto no comenzará a desarrollarse hasta que las mismas no estén totalmente disponibles. De esta manera, previamente se hará un estudio lo más adecuado y preciso posible sobre las herramientas y prestaciones a utilizar, de manera que se analice la viabilidad de cada una de las mismas.

- **Riesgo 3: La no correcta definición de los requisitos y su redefinición conforme avanza el proyecto.**
  Para llevar a cabo una correcta definición de los requisitos se utilizará una metodología consistente en obtener la información sobre el dominio del problema en primer lugar, preparar y realizar las sesiones de negociación, identificar y revisar los objetivos del sistema, identificar y revisar los requisitos de información, funcionales y no funcionales y en último lugar priorizar los objetivos y requisitos. Así mismo, todos los requisitos que se especifiquen deberán ser documentados.

- **Riesgo 4: El personal necesita tiempo extra para aprender nuevas herramientas.**
  Para evitar problemas del personal en cuanto al aprendizaje de las nuevas herramientas, se ofrecerán cursos de formación en las mismas, a los cuales se deberá de asistir obligatoriamente para aprender y afianzar la manera de trabajar con las herramientas a usar.

- **Riesgo 5: La falta de un seguimiento exacto del progreso hace que se desconozca que el proyecto esté retrasado hasta que está muy avanzado.**
  Para evitar este problema, se realizará un continuo seguimiento del progreso en cada uno de los sprint para detectar a tiempo los signos de posibles retrasos. Para ello la planificación deberá de ser supervisada y actualizada.

- **Riesgo 6: No es posible integrar los componentes desarrollados y es necesario rediseñar y repetir algunas tareas de programación.**
  Para evitar problemas de integración de componentes se invertirá el suficiente tiempo en realizar el Diagrama de Descomposición del Trabajo (WBS), y se planificará y documentará la manera en la que se realizará la integración de los diferentes componentes una vez hayan sido programados. Así mismo, durante el desarrollo del proyecto, se irán realizado diversas pruebas de integración.

- **Riesgo 7: No es posible implementar la funcionalidad deseada en el lenguaje de programación.**
  Para evitar problemas sobre la imposibilidad de implementar ciertas funcionalidades en el lenguaje de programación elegido, el estudio de viabilidad del sistema realizado de manera previa deberá de asegurar que no existan probabilidades de que dicho problema pueda aparecer.

- **Riesgo 8: Se realiza un diseño simple y preliminar que no resuelve los problemas que obligan más tarde a rediseñar y volver a codificar.**
  Para evitar realizar un mal diseño, se invertirá el suficiente tiempo en realizar el mismo, y dicho tiempo será tenido en cuenta en la planificación temporal. Toda la fase de diseño deberá ser debidamente documentada y será obligatorio el uso de patrones de diseño que contribuyan al correcto funcionamiento de la aplicación, evitando tener que volver a rediseñar y codificar módulos de la aplicación.

2.6.2.2. Resolución de riesgos

- **Riesgo 1: Planificación muy optimista.**
  Una planificación muy optimista puede llevar al proyecto a dos situaciones compatibles: falta de tiempo y falta de recursos. En el caso de que se de esta situación, lo primero que se debe estudiar es por qué se llevo a cabo esta mala planificación, con el fin de documentar el error cometido y que éste no vuelva a ocurrir. Una vez hallada la raíz del problema lo siguiente será rehacer la planificación, intentando arreglar en la medida de lo posible los retrasos causados sin volver a ser demasiado optimista.

- **Riesgo 2: Herramientas de desarrollo no disponibles o funcionamiento / prestaciones indeseadas.**
  En este caso lo primero que se hará será estudiar si el funcionamiento no deseado es fruto de un error en la aplicación o por el contrario es una característica de su funcionamiento. En el primero de los casos habrá que ponerse en contacto con el distribuidor con el fin denunciar la situación.

- **Riesgo 3: La no correcta definición de los requisitos y su redefinición conforme avanza el proyecto.**
  En el caso de la no correcta definición de los requisitos, al igual que en los otros casos, estudiaremos por qué ocurrió esto para conseguir que no vuelva a ocurrir dicha

situación. Una vez hecho esto, lo siguiente será redefinir los requisitos y verificarlos con el cliente, asegurando en la medida de lo posible que éstos son correctos.

- **Riesgo 4: El personal necesita tiempo extra para aprender nuevas herramientas.**
  El aprendizaje es un proceso difícilmente planificable, en el caso de que se produzca esta situación, se buscará una solución factible. Una de estas podría ser modificar el método de aprendizaje.

- **Riesgo 5: La falta de un seguimiento exacto del progreso hace que se desconozca que el proyecto esté retrasado hasta que está muy avanzado.**
  Lo primero que se debe estudiar es por qué no se llevo a cabo un seguimiento exacto del progreso para conseguir que no vuelva a ocurrir. Para evitar esta situación se irán anotando los progresos en el fichero *Changelog*.

- **Riesgo 6: No es posible integrar los componentes desarrollados y es necesario rediseñar y repetir algunas tareas de programación.**
  Es difícil actuar contra este riesgo, en este caso lo único que se puede hacer es documentar la situación para el futuro.

- **Riesgo 7: No es posible implementar la funcionalidad deseada en el lenguaje de programación.**
  Al igual que ocurría con el **riesgo 6**, en este caso no se puede hacer nada, solo estudiar el origen del problema y documentarlo.

- **Riesgo 8: Se realiza un diseño simple y preliminar que no resuelve los problemas que obligan más tarde a rediseñar y volver a codificar.**
  De forma especial, en este riesgo es de vital importancia evitar que vuelva a ocurrir, ya que los costes asociados a él son muy altos y puede ser evitable fácilmente.

Parte II

Desarrollo

En esta segunda parte de la memoria se describe el desarrollo del proyecto siguiendo la metodología anteriormente descrita. Va a estar estructurada según las etapas principales del proceso de ingeniería.

Capítulo 3

Análisis de requisitos

Una de las tareas más importantes en el desarrollo de aplicaciones informáticas es la *captura y documentación de requisitos*, ya que de una adecuada comprensión de los requisitos del sistema depende en gran parte el éxito del proyecto.

Para realizar la toma de requisitos de forma eficiente, he ido organizando reuniones con una artista galerista, pudiendo de esta forma recopilar y definir todos los requisitos que debe cumplir el sistema.

3.1. Catálogo de actores

Dentro del sistema he identificado los diferentes perfiles de usuarios (*Figura 3.1*) que van a trabajar y utilizar la aplicación:

- El **administrador**, encargado de la gestión de la aplicación en general, tendrá acceso a todas las funcionalidades del módulo de administración.
- El **visitante** es el usuario que navega de forma anónima por la web, tendrá solo acceso a visitar las obras contenidas en el catálogo sin poder realizar compras.
- El **cliente** es el usuario, registrado e identificado por el sistema, que accederá a la aplicación para poder consultar el catálogo de productos y realizar sus compras.

También vamos a tener un sistema externo:

- La **pasarela de pago**, la cual es proporcionada por la entidad financiera a través de la que se efectuarán las transacciones bancarias para pagar los pedidos.

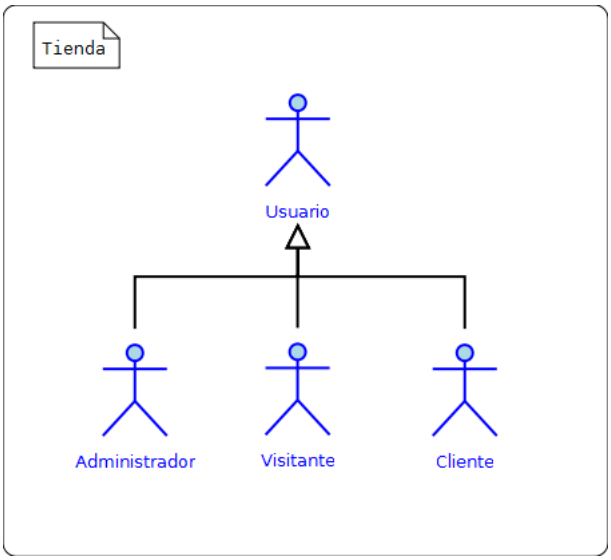


Figura 3.1: Actores del sistema

3.2. Requisitos funcionales

Voy a describir la *funcionalidad* que ofrece el sistema mediante diagramas que permiten comprender un poco más el funcionamiento del mismo. La realización de un buen análisis previo, añadirá simplicidad durante el desarrollo y mejorará el rendimiento de la aplicación.

Para ello he utilizado *UML («Lenguaje Unificado de Modelado»)*, un lenguaje de modelado utilizado en la *Ingeniería del Software* para especificar o describir métodos o procesos del sistema que se desea modelar.

3.2.1. Historias de usuario

A continuación, para identificar los requisitos funcionales, defino las *Historias de Usuario*[1] que contiene cada uno de los *Sprints* de mi aplicación:

- **Sprint 2 - Artistas**. La *Galería* poseerá información de cada uno de los *Artistas*. Estos deberán contener la siguiente información traducida: *curriculum*.
  - **Añadir artista**. El administrador de la Galería de Arte debe poder dar de alta un nuevo artista a través de un formulario, en el que rellenará sus datos.

[1]Las *Historias de Usuario* se utilizan en la *Metodología Ágil* para especificar los requisitos del sistema.

- **Introducir una imagen del artista**. El administrador deberá poder introducir una imagen del artista, la cual se mostrará a los clientes.
- **Editar artista**. El administrador deberá poder modificar los datos del artista a través de un formulario.
- **Eliminar artista**. El administrador podrá eliminar un artista de forma sencilla.
- **Listar artistas**. La interfaz del administrador necesita una página web que también funcionará como una lista de todos los artistas que posee la Galería.
- **Ver artista**. Cada artista deberá tener una página que muestre toda su información.

▪ **Sprint 3 - Colecciones** La *Galería* poseerá información de cada una de las *Colecciones de Arte* que posea. Estas deberán contener la siguiente información traducida: *nombre*.

- **Añadir colección**. El administrador debe ser capaz de dar de alta una nueva colección a través de un formulario, en el que rellenará todos los datos.
- **Editar colección**. El administrador deberá poder modificar los datos de la colección a través de un formulario.
- **Eliminar colección**. El administrador podrá eliminar una colección en cualquier momento de forma sencilla.
- **Listar colecciones**. En la interfaz del administrador deberá haber una página web que también funcionará como una lista de todas las colecciones que posee la Galería.
- **Ver colección**. Cada colección deberá tener una página que muestre toda su información.

▪ **Sprint 4 - Obras**. La *Galería* poseerá información de cada una de las *Obras de Arte* que posea en su inventario. Estas deberán contener la siguiente información traducida: *titulo, comentario*.

- **Añadir obra**. El administrador de la galería debe ser capaz de añadir nuevas obras en el inventario.
- **Introducir una imagen de la obra**. El administrador deberá poder introducir una imagen de la obra, la cual se mostrará a los clientes.
- **Editar obra**. El administrador deberá ser capaz de editar los datos de la obra a través de un formulario.
- **Eliminar obra**. El administrador debe ser capaz de eliminar una obra de arte del inventario de la Galería.
- **Ver obra de arte**. Cada obra de arte deberá tener una página en donde se muestre toda su información.

  - **Listar obras de arte**. En la interfaz del administrador deberá haber una página web que también funcionará como una lista de todas las obras de arte del inventario de la Galería.
  - **Buscar obra de arte**. El administrador deberá poder buscar obras de arte a través de diferentes campos que deberán actuar como filtros, obteniéndose un listado de todas las obras que coincidan con su búsqueda.

- **Sprint 5 - Secciones**. La Galería de Arte clasificará sus obras en diferentes *Secciones*. Estas deberán contener la siguiente información traducida: *nombre*.

  - **Añadir sección**. El administrador debe ser capaz de dar de alta una nueva sección a través de un formulario, en el que rellenará todos los datos.
  - **Editar sección**. El administrador deberá poder modificar los datos de las sección a través de un formulario.
  - **Eliminar sección**. El administrador podrá eliminar una sección en cualquier momento de forma sencilla.
  - **Listar secciones**. En la interfaz del administrador deberá haber una página web que también funcionará como una lista de todas las secciones.
  - **Ver sección**. Cada sección deberá tener una página que muestre toda su información.

- **Sprint 6 - Técnicas**. Cada una de las *Secciones* posee una serie de *Técnicas* con las que se realizan las *Obras de Arte*. Estas deberán contener la siguiente información traducida: *nombre*.

  - **Añadir técnica**. El administrador debe ser capaz de dar de alta una nueva técnica a través de un formulario, en el que rellenará todos los datos.
  - **Editar técnica**. El administrador deberá poder modificar los datos de la técnica a través de un formulario.
  - **Eliminar técnica**. El administrador podrá eliminar una técnica en cualquier momento de forma sencilla.
  - **Listar técnicas**. En la interfaz del administrador deberá haber una página web que también funcionará como una lista de todas las técnicas.
  - **Ver técnica**. Cada técnica deberá tener una página que muestre toda su información.

- **Sprint 7 - Catálogo**. El *Catálogo* servirá para que el usuario[2] pueda consultar la relación de Obras que posee la *Galería* así como la ficha técnica de cada una de ella y de su artista.

  - **Explorar catálogo de obras**. La Galería necesita una página web donde el usuario pueda ver todas las obras de arte que posee en su inventario, navegar por ellas e incluirlas en su cesta de la compra.

[2] *Usuario*: Visitante o Cliente

  - **Ficha técnica de la obra de arte**. Se necesita que el usuario pueda consultar toda la información posible de cada una de las obra de arte de la Galería.
  - **Ficha técnica del artista de la obra**. El usuario deberá poder acceder a toda la información relacionada con el artista de la obra.
  - **Obras más recientes**. El usuario deberá tener la posibilidad, en el catálogo, de obtener un listado con las obras más recientes de la Galería.
  - **Buscar obras de arte**. El usuario deberá tener la opción de buscar obras de arte a través de diferentes campos que deberán actuar como filtros, obteniéndose un listado de todas las obras que coincidan con su búsqueda.

- **Sprint 9 - Cesta de la Compra**. En la *Cesta de la Compra* el cliente podrá ir añadiendo cada una de las Obras de Arte que desee comprar. En ella se visualizará las Obras compradas, sus cantidades y el coste total de los productos.

  - **Añadir a la cesta**. El cliente deberá poder añadir la obra de arte que desee comprar en su cesta de la compra de forma fácil.
  - **Sacar de la cesta**. El cliente si se equivoca deberá tener la posibilidad de eliminar la obra de arte de su cesta de forma sencilla.
  - **Vaciar la cesta**. El cliente podrá vaciar su cesta de la compra en cualquier momento.

- **Sprint 10 - Etiquetado**. El *Etiquetado* tendrá dos fines para la *Galería de Arte*:

  - Poder tener las obras de arte clasificadas por categoría de forma que el administrador pueda acceder a dicha clasificación.
  - Recomendar obras relacionadas cuando los usuarios estén navegando por la ficha técnica de una obra.

  A continuación detallo las *Historias de Usuario* de este *Sprint*.

  - **Asignar etiquetas**. El administrador deberá poder asignar etiquetas a cada una de las obras de arte del inventario de la Galería, introduciendo una lista separadas por coma en el mismo formulario de la obra.
  - **Editar etiquetas**. El administrador podrá editar las etiquetas a través del formulario de una obra existente.
  - **Eliminar etiquetas**. El administrador podrán eliminar etiquetas a través del formulario de una obra existente.
  - **Listar etiquetas**. El administrador podrán tener acceso al listado de todas las etiquetas con las que se han clasificado las obras.
  - **Mostrar etiquetas**: En la ficha técnica de una obra deberá aparecer el conjunto de etiquetas que esta posee, pudiendo el usuario seleccionar una etiqueta para ver el listado de obras que tienen ese mismo etiquetado.

  - **Recomendar obras**: Mientras el usuario está observando la ficha técnica de una obra, el sistema debe ser capaz de recomendarle obras similares. También deberá proporcionar enlaces a las diferentes obras y etiquetas que están relacionadas con la obra actual.

- **Sprint 11 - Seguridad**. La *Galería* necesita que solo el usuario *Administrador* sea el que pueda administrar la aplicación evitando así algún que otro mal para la empresa. Para realizar la compra los visitantes deberán identificarse como clientes, o registrarse si todavía no tienen cuenta.

  - **Iniciar sesión**: La página de la *Galería* tendrá una opción para *Iniciar Sesión*, la cual redirigirá a una página de *Inicio de Sesión* donde el usuario deberá introducir su login y su clave para poder tener acceso. Una vez introducidos los datos correctos se redirigirá al catálogo donde podrá ya navegar por la web de la *Galería*. En caso de que los datos sean erróneos se le mostrará un mensaje de error.
  - **Perdida de clave**: Dentro de la página de *Inicio de Sesión* deberá aparecer una opción para obtener una nueva contraseña en caso de pérdida u olvido. Haciendo clic sobre ella se le redirigirá a una nueva página donde el usuario indicará su correo electrónico y el sistema, automáticamente, deberá enviarle un email con las instrucciones para restablecer su contraseña.
  - **Registrarse**: La página de la *Galería* también contará con una opción para que los usuarios puedan *registrarse*. Esta redirigirá a una página donde el cliente deberá rellenar un formulario para poder darse de alta. Una vez rellenado y validado, el sistema deberá enviar un email de notificación de registro al cliente y a partir de ese momento ya podrá utilizar la opción *Iniciar Sesión* sin ningún problema.
  - **Cerrar sesión**: La web también contará con esta opción para que los clientes puedan cerrar su sesión una vez finalicen.
  - **Mi cuenta**: El cliente registrado deberá poder acceder a su información y poder modificarla si fuera necesario.
  - **Listar usuarios**: El administrador podrá ver un listado de los usuarios registrados.

- **Sprint 12 - Facturación**. La *Galería* desea tener un sistema de procesamiento para los pedidos de sus clientes a través de una pasarela de pago. También ofrecerá a sus clientes la opción de pagar a través de una transferencia bancaria o ingreso en efectivo.

  - **Facturar**: Una vez que el cliente haya añadido a su cesta las obras de arte que desea comprar, deberá poder proceder a la facturación.

    Se le redirigirá a una página donde deberá introducir su información de contacto, la información para la facturación y la información de su tarjeta de crédito en caso de que desee pagar por este método. Una vez rellenado el formulario y antes de realizar el pedido, se le deberá mostrar al usuario la información

introducida para que la verifique. En este momento se inicia el flujo de trabajo de procesamiento de pedidos que implica *facturar al cliente* y *envío de las obras de arte*.

El sistema, inmediatamente, deberá enviar un correo electrónico:

- a la *Galería de Arte*, indicándole que existe una nueva solicitud de pedido
- al *cliente* como confirmación del pedido que ha realizado.

Este deberá contener la información del pedido, la información de contacto y la dirección de envío.

- **Ver los pedidos**: El administrador deberá ser capaz de ver el estado en el que se encuentran todos los pedidos (procesado, cerrado, fracasado). Los *pedidos procesados* son los que han sido facturados a los clientes pero no se han enviado todavía. Una vez hayan sido enviados deberán aparecer como cerrados.
- **Ver información de un pedido**: El administrador debe poder ver la información de un pedido en cualquier momento, donde se le mostrará la información del cliente y los detalles del pedido.
- **Cerrar pedido**: Una vez que la *Galería de Arte* haya enviado el pedido al cliente, el administrador deberá poder cerrar el pedido a través de un botón en la página que muestra la información del pedido.

- **Sprint 13 - Internacionalización**. La *Galería* desea que su página pueda verse tanto en español como en inglés, para así poder acaparar una mayor clientela.

  - **Cambiar la configuración regional**: El cliente debe ser capaz de cambiar la configuración regional a través de un vínculo en el sitio web de la *Galería*.
  - **Agregar traducción**: El administrador debe ser capaz de añadir una nueva traducción para los artistas, colecciones, obras, secciones y técnicas.
  - **Editar traducción**: El administrador debe ser capaz de modificar el texto traducido.
  - **Eliminar traducción**: El administrador debe ser capaz de eliminar el texto traducido.

- **Sprint 14 - Inicio.** La *Galería* desea tener una página de inicio donde se muestren las nuevas obras adquiridas.

- **Sprint 15 - Contactar.** La *Galería* desea tener una página desde la que el usuario pueda obtener la información de contacto y la localización de la Galería.

- **Sprint 16 - Usuarios.** La *Galería* poseerá información de cada uno de los *Usuarios* que se registre en su web.

  - **Editar usuario**. El administrador deberá ser capaz de editar los datos del usuario a través de un formulario.
  - **Eliminar usuario**. El administrador debe ser capaz de eliminar una usuario de la galería.

- **Listar usuarios**. En la interfaz del administrador deberá haber una página web que también funcionará como una lista de todas los usuarios registrados en la galería.
- **Ver usuario**. Cada usuario deberá tener una página en donde se muestre toda su información.

- **Sprint 17 - Quienes somos.** Mediante este página el usuario deberá poder acceder a la descripción de la actividad de la galería.

  - **Editar Quienes somos**. El administrador deberá ser capaz de editar el apartado de *Quienes somos* a través de un formulario.
  - **Ver Quienes somos**. Existirá una página en donde se muestre la actividad de la galería.

- **Sprint 18 - Condiciones.** Mediante este menú el usuario deberá poder acceder a las condiciones generales de la galería.

  - **Añadir condición**. El administrador debe ser capaz de añadir nuevas condiciones.
  - **Editar condición**. El administrador deberá ser capaz de editar los datos de la condición a través de un formulario.
  - **Eliminar condición**. El administrador debe ser capaz de eliminar una condición.
  - **Listar condiciones**. En la interfaz del administrador deberá haber una página web que también funcionará como una lista de todas las condiciones generales de la galería.
  - **Ver condición**.
    - *Interfaz del administrador*: Cada condición deberá tener una página en donde se muestre su información.
    - *Interfaz del usuario*: Existirá una página en la que se le mostrará al usuario todas las condiciones.

- **Sprint 19 - Panel Administración.** La *Galería* necesita una página desde la que el administrador pueda acceder a la administración de la web.

En la *Figura 4.26* podemos ver el ***Diagrama del Sistema*** donde se definen cada uno de los subsistemas (Sprint) que lo componen:

Figura 3.2: Diagrama del Sistema

3.2.2. Modelado de casos de uso

En Ingeniería del Software, un *Caso de Uso* es una técnica para la captura de requisitos potenciales de un nuevo sistema. Cada uno proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

En esta sección mostraré el estudio mediante *"Casos de Uso"* de la aplicación, donde se mostrarán las funcionalidades y los comportamientos del sistema mediante su interacción con algún agente externo[3].

3.2.2.1. Diagrama de casos de uso

A continuación se muestra el modelo inicial de *Casos de Uso* del sistema para cada uno de los *Sprint*, utilizando para ello notación *UML*.

[3] *Interacción con algún agente externo*: Desde una petición de un *actor* o bien desde la invocación desde otro *caso de uso*.

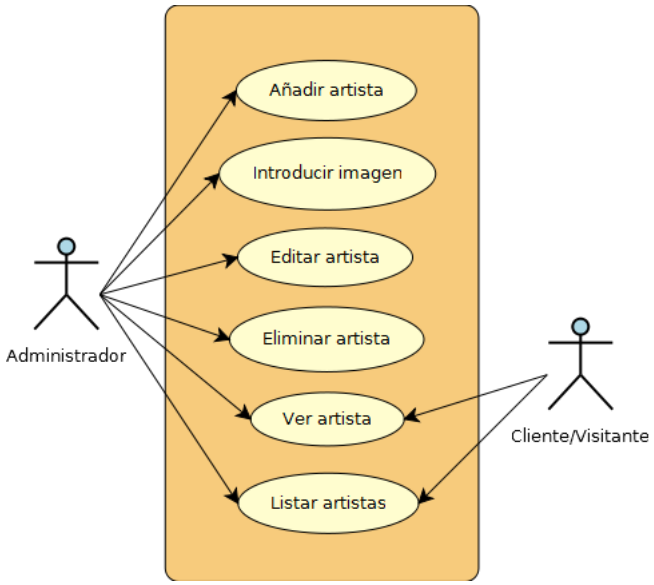


Figura 3.3: Diagrama de CU de Gestión de Artistas

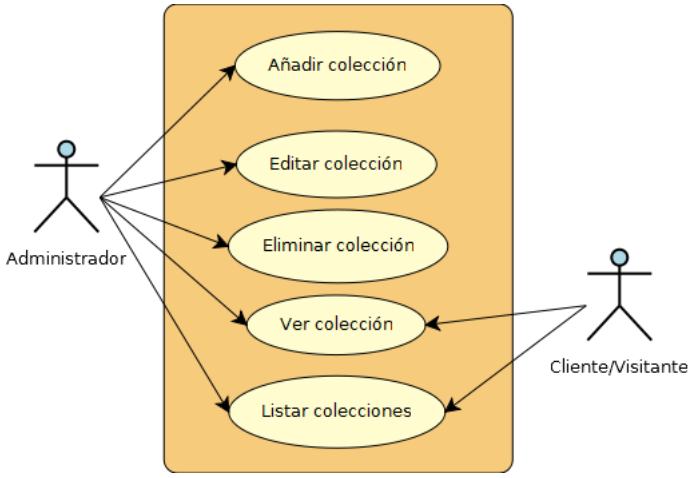


Figura 3.4: Diagrama de CU de Gestión de Colecciones

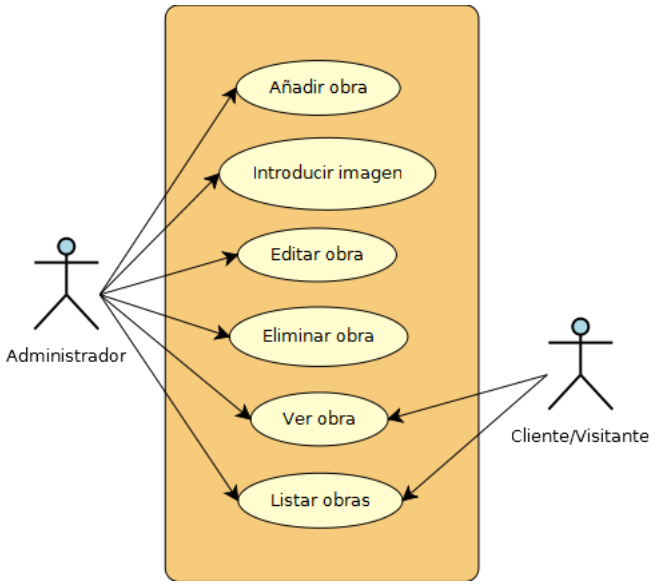


Figura 3.5: Diagrama de CU de Gestión de Obras

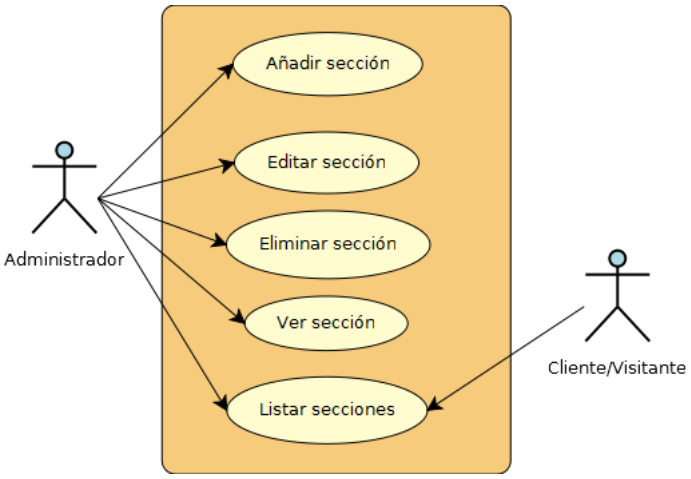


Figura 3.6: Diagrama de CU de Gestión de Secciones

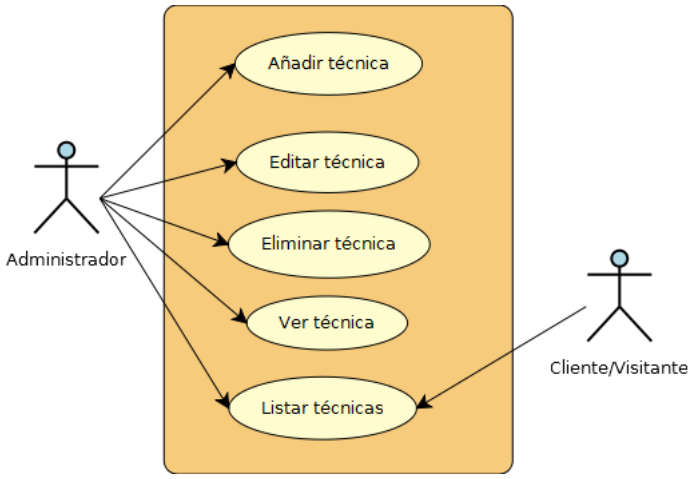


Figura 3.7: Diagrama de CU de Gestión de Técnicas

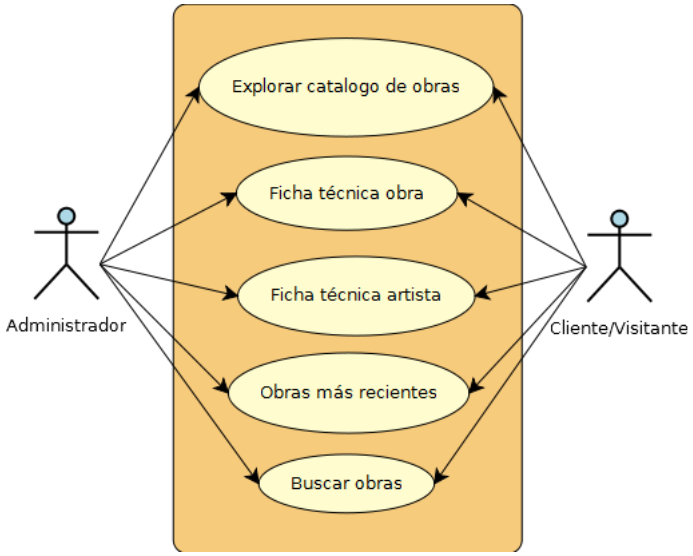


Figura 3.8: Diagrama de CU de Gestión de Catálogo

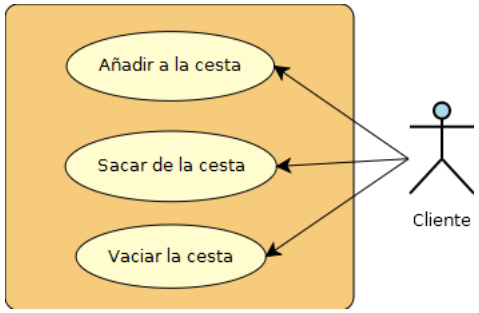


Figura 3.9: Diagrama de CU de Gestión de Cesta de la Compra

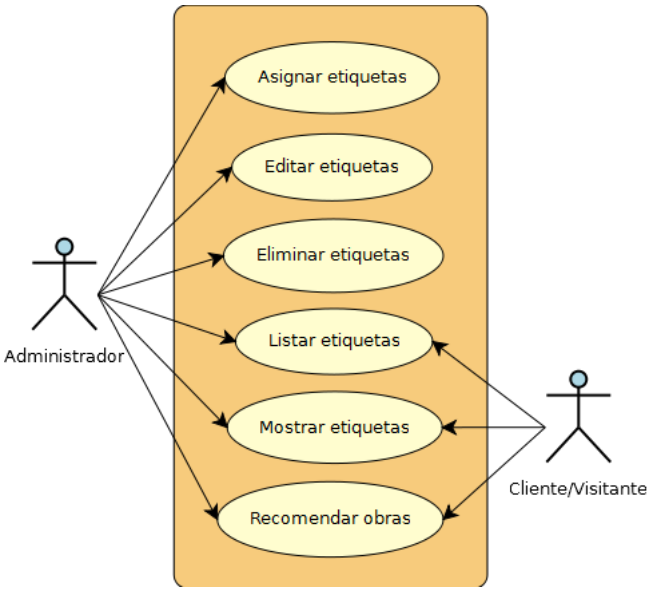


Figura 3.10: Diagrama de CU de Gestión de Etiquetas

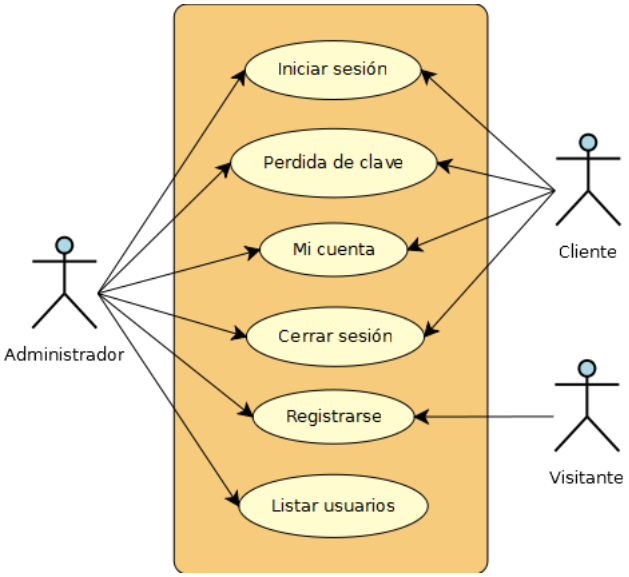


Figura 3.11: Diagrama de CU de Gestión de Seguridad

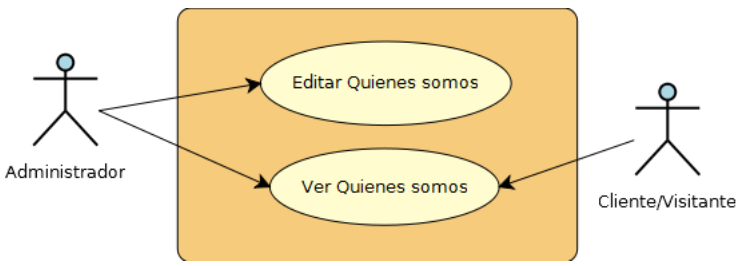


Figura 3.12: Diagrama de CU de Gestión de Quienes somos

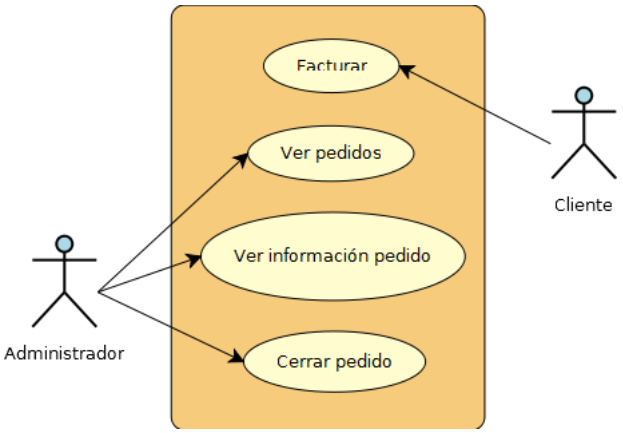


Figura 3.13: Diagrama de CU de Gestión de Facturación

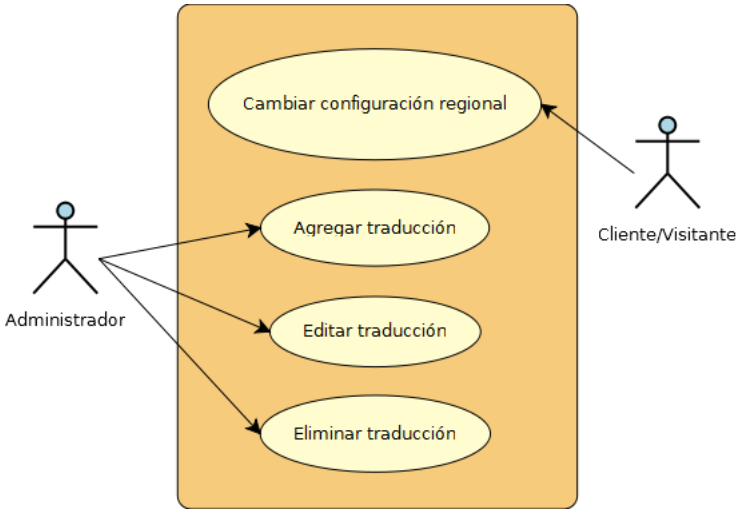


Figura 3.14: Diagrama de CU de Gestión de Internacionalización

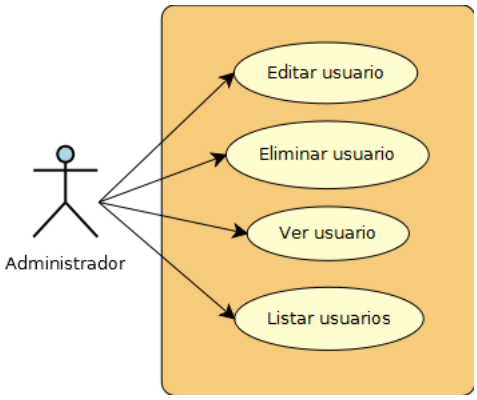


Figura 3.15: Diagrama de CU de Gestión de Usuarios

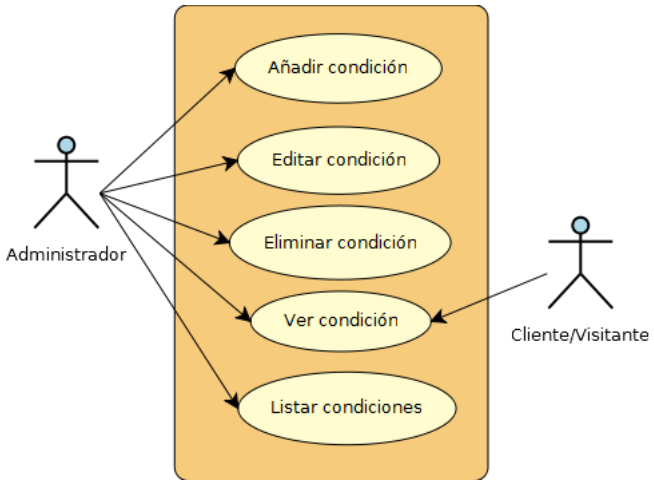


Figura 3.16: Diagrama de CU de Gestión de Condiciones

3.2.2.2. Descripción de casos de uso

En esta sección se describen los diferentes *Casos de Uso* del sistema para cada uno de los *Sprint*.

Gestión de Artistas

UC-0001	**Añadir artista.**	
Dependencias	Gestión de *artistas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *administrador solicite añadir un nuevo artista*.	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
	El *artista* no está registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona dar de alta un nuevo artista*.
	2	El sistema *muestra el formulario con los datos en blanco para que el* administrador *los rellene*.
	3	El actor administrador *introduce los datos del artista y lo envía*.
	4	El sistema *valida los datos introducidos*.
	5	El sistema *registra el alta del nuevo artista*.
Postcondición	*Se registra el alta de un nuevo artista*.	
Excepciones	**Paso**	**Acción**
	3	Si *no se han introducido los campos necesarios para añadir un nuevo artista*, el sistema *muestra un mensaje de error*, a continuación este caso de uso *queda sin efecto*.
	4	Si algún dato no ha sido validado correctamente el sistema *muestra un mensaje de error*, a continuación este caso de uso *queda sin efecto*.
	-	El administrador puede cancelar la operación en cualquier momento.

Figura 3.17: Descripción del CU Añadir Artista

UC-0003	Editar artista.	
Dependencias	Gestión de *artistas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el administrador seleccione editar un artista*.	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
	El *artista* a editar debe existir en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Administrador *accede a la opción de editar los datos de un artista*.
	2	El sistema *muestra el formulario con los datos del artista*.
	3	El actor administrador *actualiza convenientemente los datos del artista y los envía*.
	4	El sistema *registra los cambios realizados*.
Postcondición	*Se actualiza la información del artista*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento*.
	3	*Algún dato no ha sido validado correctamente*.

Figura 3.18: Descripción del CU Editar Artista

UC-0004	Eliminar artista.	
Dependencias	Gestión de *artistas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el administrador seleccione eliminar un artista*.	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
	El *artista* a eliminar debe existir.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *accede a la opción de eliminar un artista del sistema*.
	2	El sistema *pide confirmación de la operación*.
	3	El actor administrador *confirma la eliminación*.
	3	El sistema *da de baja al artista*.
Postcondición	Se elimina al *intérprete*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento*.

Figura 3.19: Descripción del CU Eliminar Artista

UC-0005	Listar artistas.	
Dependencias	Gestión de *artistas*.	
Actores	Usuario (Administrador/Cliente/Visitante)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver los artistas existentes en el sistema*	
Precondición	Debe existir al menos un *artista* registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *solicita ver un listado de los artistas existente.*
	2	El sistema *muestra una lista con todos los artistas del sistema.*
Postcondición	*Se muestra un listado de los artistas existentes.*	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento.*

Figura 3.20: Descripción del CU Listar Artistas

UC-0006	Ver artista.	
Dependencias	Gestión de *artistas*.	
Actores	Usuario (Administrador/Cliente/Visitante)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver la información de un artista existente en el sistema.*	
Precondición	El *artista* a visualizar debe existir en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *solicita ver la información existente de un artista concreto.*
	2	El sistema *muestra la información asociada a dicho artista.*
Postcondición	*Se muestra la información almacenada de un artista.*	

Figura 3.21: Descripción del CU Ver Artista

UC-0002	Introducir imagen del artista.	
Dependencias	Gestión de *artistas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *administrador solicite añadir una imagen del artista.*	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
	El *artista* debe estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *desea añadir una imagen del artista.*
	2	El sistema *muestra en el formulario la opción para introducir la imagen.*
	3	El actor administrador *introduce la imagen del artista y lo envía.*
	4	El sistema *registra la imagen del artista.*
Postcondición	*Se registra la imagen del artista.*	
Excepciones	**Paso**	**Acción**
	-	El administrador puede cancelar la operación en cualquier momento.

Figura 3.22: Descripción del CU Introducir imagen del Artista

Las descripciones para los CU de *Gestión de Colecciones, Gestión de Obras, Gestión de Secciones, Gestión de Técnicas, Gestión de Usuarios, Gestión de Quienes somos y Gestión de Condiciones* son similares a la *Gestión de Artistas*.

Gestión de Catálogo

UC-0010	**Buscar Obras.**	
Dependencias	Gestión de *catálogo*.	
Actores	Usuario (Administrador/Cliente/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite buscar obras*.	
Precondición	Debe existir al menos una *obra* registrada en el sistema.	
Secuencia normal	**Paso**	**Secuencia normal**
	1	El actor *usuario selecciona la opción de realizar una búsqueda en el catálogo.*
	2	El sistema *muestra un formulario para insertar el valor con el que queremos realizar la búsqueda.*
	3	El *usuario introduce el valor con el que desea realizar la búsqueda y le da al botón buscar.*
	4	El sistema *muestra las obras que coinciden con el valor insertado para la búsqueda.*
Postcondición	*Se muestra un listado de las obras que coinciden con el valor de la búsqueda.*	
Excepciones	**Paso**	**Acción**
	-	El *usuario* puede cancelar la operación en cualquier momento.
	4	No existe ninguna *obra* que cumpla con el valor de la búsqueda.

Figura 3.23: Descripción del CU Buscar Obras

UC-0007	**Explorar catálogo de obras.**	
Dependencias	Gestión de *catálogo*.	
Actores	Usuario (Administrador/Cliente/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver el catálogo de las obras de arte.*	
Precondición	Debe existir al menos una *obra* registrada en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor *usuario accede a la opción del catálogo de obras*.
	2	El sistema *muestra el catálogo de obras*.
Postcondición	*Se muestra el catálogo de obras.*	
Excepciones	**Paso**	**Acción**
	-	El *usuario* puede cancelar la operación en cualquier momento.

Figura 3.24: Descripción del CU Explorar catálogo de Obras

UC-0008	Ficha técnica obra.	
Dependencias	Gestión de *catálogo*.	
Actores	Usuario (Administrador/Cliente/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver la ficha técnica de una obra*.	
Precondición	La *obra* debe estar registrada en el sistema.	
Secuencia normal	**Paso**	**Secuencia normal**
	1	El actor *usuario selecciona la obra de la que desea ver la ficha técnica*.
	2	El sistema *muestra la ficha técnica de la obra*.
Postcondición	*Se muestra la ficha técnica de la obra*.	
Excepciones	**Paso**	**Acción**
	-	El *usuario* puede cancelar la operación en cualquier momento.

Figura 3.25: Descripción del CU Ficha técnica de la Obra

UC-0008	Ficha técnica artista.	
Dependencias	Gestión de *catálogo*.	
Actores	Usuario (Administrador/Cliente/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver la ficha técnica del artista*.	
Precondición	El *artista* debe estar registrado en el sistema.	
Secuencia normal	**Paso**	**Secuencia normal**
	1	El actor *usuario selecciona el artista de la que desea ver la ficha técnica*.
	2	El sistema *muestra la ficha técnica del artista*.
Postcondición	*Se muestra la ficha técnica de la artista*.	
Excepciones	**Paso**	**Acción**
	-	El *usuario* puede cancelar la operación en cualquier momento.

Figura 3.26: Descripción del CU Ficha técnica del artista

UC-0009	Obras más recientes.	
Dependencias	Gestión de *catálogo*.	
Actores	Usuario (Administrador/Cliente/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver las obras más recientes*.	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Secuencia normal**
	1	El actor *usuario selecciona ver las obras más recientes*.
	2	El sistema *muestra un listado de las obras más recientes*.
Postcondición	*Se muestra un listado de las obras más recientes*.	
Excepciones	**Paso**	**Acción**
	-	El *usuario* puede cancelar la operación en cualquier momento.

Figura 3.27: Descripción del CU Obras más recientes

Gestión de Cesta de la Compra

UC-0011	**Añadir a la cesta.**	
Dependencias	Gestión de *cesta de la compra*.	
Actores	Cliente	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el cliente *solicite añadir una obra de arte* a la cesta.	
Precondición	El cliente debe de estar debidamente autentificado y autorizado.	
	La *obra* a insertar debe estar registrada en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor cliente *selecciona una determinada obra del catálogo y pulsa sobre el botón de insertar en la cesta.*
	2	El sistema *muestra un mensaje de confirmación.*
	3	El actor cliente *confirma la acción de insertar la obra en la cesta.*
	4	El sistema *inserta la obra en la cesta e incrementa el coste.*
	5	El actor cliente *puede volver a insertar otra obra en la cesta.*
Postcondición	*Se inserta una obra en la cesta de la compra.*	
Excepciones	**Paso**	**Acción**
	-	El cliente puede cancelar la operación en cualquier momento.

Figura 3.28: Descripción del CU Añadir a la Cesta

UC-0012	**Sacar de la cesta.**	
Dependencias	Gestión de *cesta de la compra*.	
Actores	Cliente	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el cliente *solicite sacar una obra de la cesta.*	
Precondición	El cliente debe de estar debidamente autentificado y autorizado.	
	La cesta debe existir.	
	Debe existir al menos una *obra* insertada en la cesta.	
Secuencia normal	**Paso**	**Secuencia normal**
	1	El actor cliente *selecciona la obra que desea eliminar de la cesta y pulsa sobre el botón de eliminar.*
	2	El sistema *muestra un mensaje de confirmación.*
	3	El actor cliente *confirma la acción de eliminar la obra de la cesta.*
	4	El sistema *elimina la obra de la cesta y actualiza el coste.*
Postcondición	*Se elimina una obra en la cesta de la compra.*	
Excepciones	**Paso**	**Acción**
	-	El cliente puede cancelar la operación en cualquier momento.

Figura 3.29: Descripción del CU Eliminar de la Cesta

UC-0013	Vaciar la cesta.	
Dependencias	Gestión de *cesta de la compra*.	
Actores	Cliente	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el cliente *solicite vaciar la cesta*.	
Precondición	El cliente debe de estar debidamente autentificado y autorizado.	
	La cesta debe existir.	
	Debe existir al menos una *obra* insertada en la cesta.	
Secuencia normal	**Paso**	**Secuencia normal**
	1	El actor cliente *selecciona la opción de vaciar el carrito.*
	2	El sistema *muestra un mensaje de confirmación.*
	3	El actor cliente *confirma la acción de vaciar la cesta.*
	4	El sistema *elimina todas las obras de la cesta y la elimina.*
Postcondición	*Se vacía la cesta de la compra.*	
Excepciones	**Paso**	**Acción**
	-	El cliente puede cancelar la operación en cualquier momento.

Figura 3.30: Descripción del CU Vaciar la Cesta

Gestión de Etiquetas

UC-0013	Asignar etiquetas.	
Dependencias	Gestión de *etiquetas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *administrador solicite asignar etiquetas a una obra*.	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *realiza el CU de añadir obra*.
	2	El actor administrador *introduce las etiquetas que quiere asociar a una determinada obra y valida la acción.*
	3	El sistema *registra las etiquetas asociadas a la obra*.
Postcondición	Se registra las etiquetas asociadas a una *obra*.	
Excepciones	**Paso**	**Acción**
	-	El administrador puede *cancelar la operación en cualquier momento.*

Figura 3.31: Descripción del CU Asignar Etiquetas

UC-0014	**Editar etiquetas.**	
Dependencias	Gestión de *etiquetas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el administrador seleccione editar* etiquetas de una *obra*.	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
	El *obra* de la que se quiere editar las etiquetas debe existir en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona la obra a la que le quiere editar las etiquetas.*
	2	El sistema *muestra el formulario con los datos de la obra.*
	3	El actor administrador *actualiza convenientemente el valor de las etiquetas y valida la acción.*
	4	El sistema *registra los cambios realizados.*
Postcondición	Se actualiza la información de la *obra*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento.*

Figura 3.32: Descripción del CU Editar Etiquetas

UC-0015	**Eliminar etiquetas.**	
Dependencias	Gestión de *etiquetas*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el administrador seleccione eliminar etiquetas de una obra*.	
Precondición	El *administrador* debe de estar debidamente autentificado y autorizado.	
	El *obra* de la que se quiere eliminar las etiquetas debe existir en el sistema	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona la obra a la que le quiere eliminar las etiquetas.*
	2	El sistema *muestra el formulario con los datos de la obra.*
	3	El actor administrador *elimina el valor de las etiquetas y valida la acción.*
	3	El sistema *registra los cambios realizados.*
Postcondición	Se eliminan las etiquetas de una *obra*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento.*

Figura 3.33: Descripción del CU Eliminar Etiquetas

UC-0016	Listar etiquetas.	
Dependencias	Gestión de *etiquetas*.	
Actores	Usuario (Administrador/Cliente/Visitante)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver un listado de las etiquetas existentes en el sistema*	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *solicita ver un listado de las etiquetas existentes en el sistema.*
	2	El sistema *muestra una lista con todas las etiquetas asociadas a las obras registradas en el sistema.*
	3	El actor usuario *podrá seleccionar una etiqueta y validar la acción.*
	4	El sistema *muestra una lista con las obras que tienen en común la etiqueta seleccionada.*
Postcondición	*Se muestra un listado de las etiquetas asociadas a las obras.*	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento.*
	2	No existen etiquetas asociadas a las *obras* registradas en el sistema. El sistema *mostrará el listado vacio.*

Figura 3.34: Descripción del CU Listar Etiquetas

UC-0017	Mostrar etiquetas.	
Dependencias	Gestión de *etiquetas*.	
Actores	Usuario (Administrador/Cliente/Visitante)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver las etiquetas de una obra.*	
Precondición	La *obra* debe existir en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *solicita ver las etiquetas asociadas a una obra.*
	2	El sistema *muestra la ficha técnica de la obra, incluidas las etiquetas.*
	3	El actor usuario *podrá seleccionar una etiqueta y validar la acción.*
	4	El sistema *muestra una lista con las obras que tienen en común la etiqueta seleccionada.*
Postcondición	*Se muestra un listado de las etiquetas asociadas a las obras.*	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento.*
	2	No existen etiquetas asociadas a dicha *obra*. El sistema *mostrará el campo vacío.*

Figura 3.35: Descripción del CU Mostrar Etiquetas

UC-0018	Recomendar obras.	
Dependencias	Gestión de *etiquetas*.	
Actores	Usuario (Administrador/Cliente/Visitante)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite ver las obras relacionadas con otra obra*.	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *solicita ver la información de una determinada obra*.
	2	El sistema *muestra la ficha técnica de la obra*.
	3	El sistema *muestra un listado de las obras y las etiquetas recomendadas a partir de la obra solicitada*.
Postcondición	*Se muestra un listado de las obras y etiquetas recomendadas a partir de una obra*.	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento*.
	2	No existen recomendaciones asociadas a dicha *obra*.

Figura 3.36: Descripción del CU Recomendar Obras

Gestión de Seguridad

UC-0019	Iniciar Sesión.	
Dependencias	Gestión de *seguridad*.	
Actores	Usuario (Administrador/Cliente).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite iniciar una sesión*.	
Precondición	El *usuario* debe de estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor *usuario selecciona la acción Iniciar Sesión*.
	2	El sistema *muestra un formulario donde se deberá introducir el login y la contraseña*.
	3	El actor *usuario introduce la información y valida la acción*.
	4	El sistema *valida la acción*.
	5	El sistema *abre la sesión al usuario y lo redirige al catálogo de obras de arte*.
Postcondición	Se registra las etiquetas asociadas a una *obra*.	
Excepciones	**Paso**	**Acción**
	-	El *usuario* puede *cancelar la operación en cualquier momento*.
	5	La autenticación del *usuario no ha sido correcta*. El sistema *muestra un mensaje de error y redirigirá de nuevo a la ventana de Iniciar Sesión*.

Figura 3.37: Descripción del CU Iniciar Sesión

UC-0020	Pérdida de clave.	
Dependencias	Gestión de *seguridad*.	
Actores	Usuario (Administrador/Cliente).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario seleccione ¿Olvidó su contraseña?*	
Precondición	El usuario debe de estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor *usuario selecciona la acción ¿Olvidó su contraseña?*
	2	El sistema *muestra el formulario en el que solicita el email del usuario.*
	3	El actor *usuario introduce su email en el formulario y selecciona restablecer contraseña.*
	4	El sistema *valida la información recibida.*
	5	El sistema *envía un email al usuario indicándole como restaurar su contraseña.*
	6	El actor *usuario seguirá las indicaciones del email recibido.*
Postcondición	Se restaura la contraseña.	
Excepciones	**Paso**	**Acción**
	-	El *usuario puede cancelar la operación en cualquier momento.*
	4	No existe ningún usuario con esa dirección de correo electrónico. El sistema muestra un mensaje de error y redirigirá de nuevo a la ventana para que el *usuario* vuelva a introducir un email válido.

Figura 3.38: Descripción del CU Pérdida de Clave

UC-0021	Mi cuenta.	
Dependencias	Gestión de *seguridad*.	
Actores	Usuario (Administrador/Cliente)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario solicite entrar en su cuenta.*	
Precondición	El usuario debe de estar registrado en el sistema.	
	El usuario debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *selecciona la acción Mi cuenta.*
	2	El sistema *muestra la información del* usuario.
	3	El actor usuario *podrá editar su información accediendo a dicha acción.*
	4	El sistema *muestra un formulario conteniendo los datos del* usuario.
	5	El actor usuario *podrá actualizar su información y validar la acción.*
	6	El sistema *valida la información y la actualizará.*
Postcondición	Se muestra la información del *usuario*.	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento.*
	5	Algún dato no ha sido validado correctamente. El sistema *muestra un mensaje de error y redirige otra vez al formulario.*

Figura 3.39: Descripción del CU Mi Cuenta

UC-0022	**Cerrar sesión.**	
Dependencias	Gestión de *seguridad*.	
Actores	Usuario (Administrador/Cliente)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *usuario cerrar la sesión*.	
Precondición	El usuario debe de estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *selecciona la acción Salir*.
	2	El sistema *cierra la sesión del usuario*.
Postcondición	Se cierra la sesión del *usuario*.	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento*.

Figura 3.40: Descripción del CU Cerrar Sesión

UC-0023	**Registrarse.**	
Dependencias	Gestión de *seguridad*.	
Actores	Usuario (Administrador/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario seleccione registrarse en la tienda*.	
Precondición	El *usuario no* debe de estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor *usuario selecciona la acción de registrarse en la tienda*.
	2	El sistema *muestra un formulario para que el usuario lo rellene con los datos solicitados*.
	3	El actor administrador *rellena la información solicitada y valida la acción*.
	4	El sistema *valida la información insertada*.
	5	El sistema *registra un nuevo usuario y le envía un email de notificación de registro*.
Postcondición	Se registra un nuevo *usuario*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento*.
	4	Algún dato no ha sido validado correctamente. El sistema *muestra un mensaje de error y redirige otra vez al formulario*.

Figura 3.41: Descripción del CU Registrarse

UC-0024	Listar usuarios.	
Dependencias	Gestión de *seguridad*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el administrador *solicite ver el listado de los usuarios registrados*.	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *solicita ver el listado de usuarios registrados*.
	2	El sistema *muestra el listado de usuarios registrados*.
Postcondición	*Se muestra un listado de los usuarios registrados en el sistema.*	
Excepciones	**Paso**	**Acción**
	-	El Usuario *puede cancelar la operación en cualquier momento.*

Figura 3.42: Descripción del CU Listar Usuarios

Gestión de Facturación

UC-0025	Facturar.	
Dependencias	Gestión de *facturación*.	
Actores	Cliente.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el *cliente solicite facturar su pedido*.	
Precondición	El *cliente* debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor *cliente decide facturar su pedido y valida la acción.*
	2	El sistema *muestra un formulario solicitando los datos para la facturación del pedido y envío.*
	3	El actor *cliente introduce la información y valida la acción.*
	4	El sistema *valida los datos y muestra la información introducida para que sea aceptada por el cliente*.
	5	El actor *cliente confirma los datos realiza el pedido*.
	6	El sistema *factura el pedido y lo registra.*
	7	El sistema *envía un email a la Galería de Arte y al cliente de confirmación de pedido*.
Postcondición	Se factura un *pedido*.	
Excepciones	**Paso**	**Acción**
	-	El *cliente* puede *cancelar la operación en cualquier momento.*
	4	Algún dato no ha sido validado correctamente. El sistema *muestra un mensaje de error y redirige otra vez al formulario.*

Figura 3.43: Descripción del CU Facturar

UC-0026	**Ver pedidos.**	
Dependencias	Gestión de *facturación*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el* administrador *seleccione ver el estado de los* pedidos *registrados*.	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona ver los* pedidos *registrados*.
	2	El sistema *muestra la lista de los* pedidos *registrados informando del estado de estos*.
Postcondición	Se muestra un listado de los *pedidos* junto con el estado de estos.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento*.
	2	No existe ningún pedido registrado en el sistema. El sistema *mostrará la lista vacía*.

Figura 3.44: Descripción del CU Ver los Pedidos

UC-0027	**Ver información pedido.**	
Dependencias	Gestión de *facturación*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el* administrador *seleccione ver la información de un* pedido *concreto*.	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
	El *pedido* debe estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor *usuario selecciona ver la información de un* pedido.
	2	El sistema *muestra la información del* pedido.
Postcondición	Se muestra la información de un *pedido*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento*.

Figura 3.45: Descripción del CU Ver información de un Pedido

UC-0028	Cerrar pedido.	
Dependencias	Gestión de *facturación*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el* administrador *seleccione cerrar un* pedido.	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
	El *pedido* debe estar registrado en el sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona el pedido que se ha enviado y desea cerrar.*
	2	El sistema *muestra los detalles del* pedido.
	3	El actor administrador *valida la acción.*
	4	El sistema *cambia el estado del pedido seleccionado a cerrado.*
Postcondición	Se cierra la sesión del *usuario*.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento.*

Figura 3.46: Descripción del CU Cerrar Pedido

Gestión de Internacionalización

UC-0029	Cambiar configuración regional.	
Dependencias	Gestión de *internacionalización*.	
Actores	Usuario (Cliente/Visitante).	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario *solicite cambiar la configuración regional.*	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor usuario *selecciona cambiar la configuración regional.*
	2	El sistema *muestra la página en la configuración seleccionada.*
Postcondición	Se cambia la configuración regional	

Figura 3.47: Descripción del CU Cambiar la configuración regional

UC-0030	**Agregar traducción.**	
Dependencias	Gestión de *internacionalización*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el* administrador *seleccione agregar una traducción.*	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona añadir una traducción.*
	2	El sistema *muestra el formulario donde introducir la traducción.*
	3	El actor administrador *introduce la traducción en el formulario y valida la acción.*
	4	El sistema *muestra la traducción introducida.*
Postcondición	Se agrega una traducción.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento.*

Figura 3.48: Descripción del CU Agregar traducción

UC-0031	**Editar traducción.**	
Dependencias	Gestión de *internacionalización*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el* administrador *seleccione editar una traducción.*	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona editar una traducción.*
	2	El sistema *muestra el formulario con la traducción.*
	3	El actor administrador *introduce las modificaciones en la traducción y valida la acción.*
	4	El sistema *muestra la traducción introducida.*
Postcondición	Se edita una traducción.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento.*

Figura 3.49: Descripción del CU Editar traducción

UC-0032	Eliminar traducción.	
Dependencias	Gestión de *internacionalización*.	
Actores	Administrador.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el* administrador *seleccione eliminar una traducción.*	
Precondición	El administrador debe de estar debidamente autentificado y autorizado.	
Secuencia normal	**Paso**	**Acción**
	1	El actor administrador *selecciona eliminar una traducción.*
	2	El sistema *muestra el formulario con la traducción.*
	3	El actor administrador *elimina la traducción y valida la acción.*
	4	El sistema *elimina la traducción.*
Postcondición	Se elimina la traducción.	
Excepciones	**Paso**	**Acción**
	-	El administrador *puede cancelar la operación en cualquier momento.*

Figura 3.50: Descripción del CU Eliminar traducción

3.3. Requisitos de Información

Éstos especifican la información que debe almacenar el sistema para cumplir con los objetivos.

Los *requisitos de información* del sistema son:

IRQ-0001	Información sobre los artistas
Objetivo asociado	**OBJ-0001 – Gestión de artistas** **OBJ-0011 – Gestión de internacionalización**
Descripción	El sistema deberá almacenar la información correspondiente a cada artista suministrados por la empresa. En concreto:
Datos específicos	• Identificador • Alias • Nombre • Curriculum • Foto
Estabilidad	Alta
Comentarios	-

Figura 3.51: Requisito de información sobre artistas

IRQ-0002	Información sobre las colecciones
Objetivo asociado	**OBJ-0002 – Gestión de colecciones** **OBJ-0011 – Gestión de internacionalización**
Descripción	El sistema deberá almacenar la información correspondiente a cada una de las colecciones suministradas por la empresa. En concreto:
Datos específicos	• Identificador • Nombre • Descripción
Estabilidad	Alta
Comentarios	-

Figura 3.52: Requisito de información sobre colecciones

IRQ-0003	Información sobre las obras de arte
Objetivo asociado	**OBJ-0003 – Gestión de obras** **OBJ-0006 – Gestión del catálogo** **OBJ-0011 – Gestión de internacionalización**
Descripción	El sistema deberá almacenar la información correspondiente a cada una de las obras de arte suministradas por la empresa. En concreto:
Datos específicos	• Identificador • Referencia • Título • Año de realización • Medidas • Comentario • Precio • Imagen de la obra
Estabilidad	Alta
Comentarios	-

Figura 3.53: Requisito de información sobre obras

IRQ-0004	Información sobre las secciones
Objetivo asociado	**OBJ-0004 – Gestión de secciones** **OBJ-0011 – Gestión de internacionalización**
Descripción	El sistema deberá almacenar la información correspondiente a cada una de las secciones suministradas por la empresa. En concreto:
Datos específicos	• Identificador • Nombre
Estabilidad	Alta
Comentarios	-

Figura 3.54: Requisito de información sobre secciones

IRQ-0005	Información sobre las técnicas
Objetivo asociado	OBJ-0005 – Gestión de técnicas OBJ-0011 – Gestión de internacionalización
Descripción	El sistema deberá almacenar la información correspondiente a cada una de las técnicas suministradas por la empresa. En concreto:
Datos específicos	• Identificador • Nombre
Estabilidad	Alta
Comentarios	-

Figura 3.55: Requisito de información sobre técnicas

IRQ-0006	Información sobre el carrito
Objetivo asociado	OBJ-0007 – Gestión de la cesta de la compra
Descripción	El sistema deberá almacenar la información correspondiente a cada uno de los carritos de los clientes. En concreto:
Datos específicos	• Identificador • Cantidad • Precio
Estabilidad	Alta
Comentarios	-

Figura 3.56: Requisito de información sobre cesta de la compra

IRQ-0009	Información sobre las etiquetas de las obras
Objetivo asociado	OBJ-0008 – Gestión de etiquetas
Descripción	El sistema deberá almacenar la información correspondiente a cada una de las etiquetas de las obras. En concreto:
Datos específicos	• Identificador • Nombre
Estabilidad	Alta
Comentarios	-

Figura 3.57: Requisito de información sobre etiquetas

IRQ-0007	Información sobre los pedidos
Objetivo asociado	OBJ-0010 – Gestión de facturación
Descripción	El sistema deberá almacenar la información correspondiente a cada uno de los pedidos realizados por los clientes. En concreto:
Datos específicos	• Identificador • Cantidad • Precio • Email • Teléfono • Nombre de entrega • Apellidos de entrega • Dirección de entrega • Ciudad de entrega • CP de entrega • País de entrega • Estado del pedido • Ip del cliente • Mensaje de error • Forma de pago
Estabilidad	Alta
Comentarios	-

Figura 3.58: Requisito de información sobre pedidos

IRQ-0008	Información sobre los usuarios
Objetivo asociado	OBJ-0012 – Gestión de usuarios OBJ-0009 – Gestión de seguridad
Descripción	El sistema deberá almacenar la información correspondiente a cada uno de los usuarios (clientes) suministrados por la empresa. En concreto:
Datos específicos	• Identificador • Login • Email • Password • Nombre • Apellidos • Dirección • Población • Ciudad • CP • País • Teléfono
Estabilidad	Alta
Comentarios	-

Figura 3.59: Requisito de información sobre usuarios

IRQ-0010	Información sobre quienes somos
Objetivo asociado	OBJ-0013 – Gestión de quienes somos OBJ-0011 – Gestión de internacionalización
Descripción	El sistema deberá almacenar la información correspondiente a quienes somos. En concreto:
Datos específicos	• Identificador • Título • Texto • Imagen de la galería
Estabilidad	Alta
Comentarios	-

Figura 3.60: Requisito de información sobre quienes somos

IRQ-0011	Información sobre las condiciones
Objetivo asociado	OBJ-0014 – Gestión de las condiciones OBJ-0011 – Gestión de internacionalización
Descripción	El sistema deberá almacenar la información correspondiente a cada una de las condiciones generales de la galería. En concreto:
Datos específicos	• Identificador • Título • Condición
Estabilidad	Alta
Comentarios	-

Figura 3.61: Requisito de información sobre condiciones

Modelado conceptual de datos

El **modelo conceptual de datos** se obtiene a partir de la *especificación de requerimientos*. Éste está orientado a representar los elementos que intervienen en el sistema y sus relaciones.

Comenzaré describiendo cada uno de los objetos que van a formar parte de él y finalizaré representado este modelo mediante el *Diagrama E/R*.

Tipos de entidades

Las entidades representan cada uno de los elementos contenidos en el modelo conceptual de datos.

Entidad	Descripción
admin_artista	Artistas de la Galería.
admin_coleccions	Colecciones de las obras.
admin_obras	Obras de arte.
admin_seccions	Secciones en que se dividen las obras de arte.
admin_tecnicas	Técnicas empleadas para realizar las obras.
carritos	Cesta de la compra.
carrito_items	Cada una de las obras insertadas en la cesta de la compra.
tags	Etiquetas para clasificar las obras.
users	Usuarios de la aplicación.
user_sessions	Sesiones de usuarios.
pedidos	Pedidos realizados por los usuarios.
pedido_items	Cada una de las obras que forman el pedido.
admin_nosotros	Información sobre la actividad de la Galería.
admin_condiciones	Condiciones generales.

Tabla 3.1: Entidades del sistema

Tipos de relaciones

La tabla 3.2 recoge las relaciones establecidas entre las distintas entidades del modelo conceptual de datos.

Diagrama Entidad/Relación

El *diagrama* o *modelo Entidad/Relación* es una herramienta para el modelado de datos de un sistema de información.

El diagrama E/R, mostrado en la *Figura 3.62*, representa el modelo de datos implementado en mi proyecto. En él se determinan las entidades, sus atributos y las relaciones existentes.

Diagrama conceptual de clases

El *diagrama de clases* es el diagrama principal para el análisis y el diseño. Este describe gráficamente la estructura de un sistema mostrando sus clases, atributos y las relaciones existentes entre ellos. En la *figura 3.63* se puede observar el diagrama de clases, diseñado con UML, para la aplicación:

Relación	**Descripción**	**Entidades**
Crea	Asocia al artista con la obra. Un artista puede crear varias obras. Una obra pertenece a un único artista.	admin_artista [1:N] admin_obras [1:1]
Formada_por	Asocia la colección con la obra. Es posible que exista obra que no tengan colección. Una colección puede tener varias obras.	admin_obras [0:1] admin_coleccions [1:N]
Contiene	Asocia la sección con la obra. A una sección pueden pertenecer varias obras. Una obra solo pertenece a una única sección.	admin_seccions [1:N] admin_obras [1:1]
Posee	Asocia la técnica con la obra. Una técnica puede pertenecer a varias obras. La obra se realiza con una única técnica.	admin_seccions [1:N] admin_obras [1:1]
Incluye	Asocia la sección con la técnica. Una sección se compone de varias técnicas. Una técnica solo pertenece a una única sección.	admin_seccions [1:N] admin_tecnicas [1:1]
Clasifica	Asocia la etiqueta con la obra. Una etiqueta puede pertenecer a varias obras. Una obra puede no tener etiquetas.	tags [1:N] admin_obras [0:N]
Introduce	Asocia la línea del pedido con la obra. Una línea de pedidos sólo pertenece a una obra. Una obra puede tener varias líneas de pedidos.	pedido_items [1:1] admin_obras [1:N]
Consta	Asocia la línea del pedido con el pedido. Un pedido puede contener varias líneas de pedido. Una línea de pedidos sólo pertenece a un pedido.	pedido_items [1:1] pedidos [1:N]
Introduce	Asocia la línea del carrito con la obra. Una línea de carrito pertenece a una única obra. Una obra puede existir en varias líneas de pedido.	carrito_items [1:1] admin_obras [1:N]
Consta	Asocia la línea del carrito con el carrito. Una línea de pedidos pertenece a un único carrito. Un carrito puede contener varias líneas de pedido.	carrito_items [1:1] carritos [1:N]
Genera	Asocia el carrito con el pedido. Un carrito solo puede generar un pedido. Un pedido solo puede ser generado por un carrito.	carritos [1:1] pedidos [1:1]
Realiza	Asocia el pedido con el usuario. Es posible que no exista pedidos para un usuario. Un pedido solo puede ser realizado por un usuario.	pedidos [1:1] users [0:N]
Tiene	Asocia el usuario con la sesión de usuario. Un usuario solo puede tener una sesión. Una sesión solo pertenece a un usuario.	users [1:1] user_sessions [1:1]
Redacta	Asocia el usuario con las condiciones. Un usuario puede redactar varias condiciones. Condición solo puede ser redactada por un usuario.	users [0:N] admin_condiciones [1:1]
Escribe	Asocia el usuario con nosotros. Un usuario solo puede escribir un nosotros. Un nosotros solo puede ser escrito por un usuario.	users [1:1] admin_nosotros [1:1]

Tabla 3.2: Relaciones del sistema

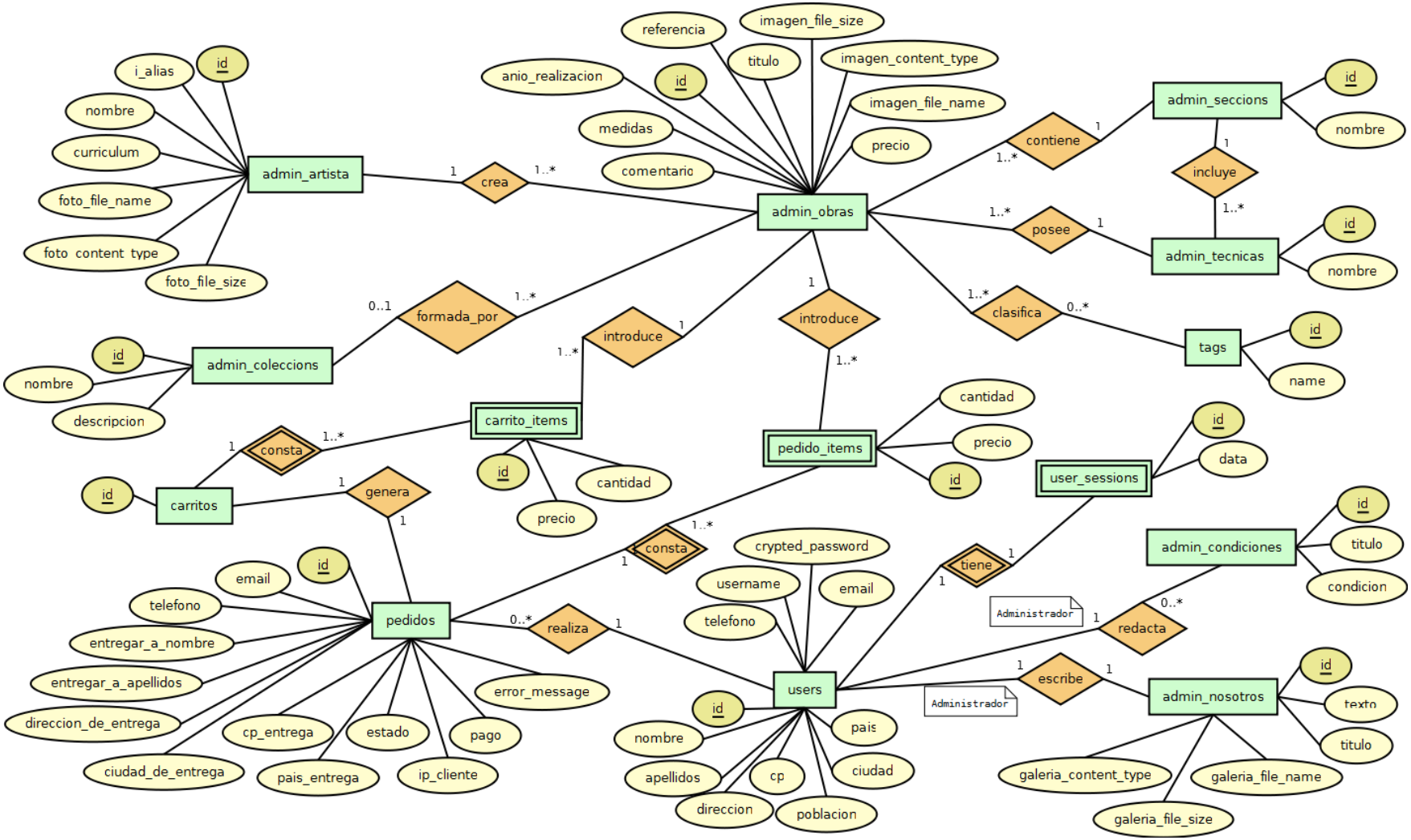


Figura 3.62: Diagrama Entidad/Relación

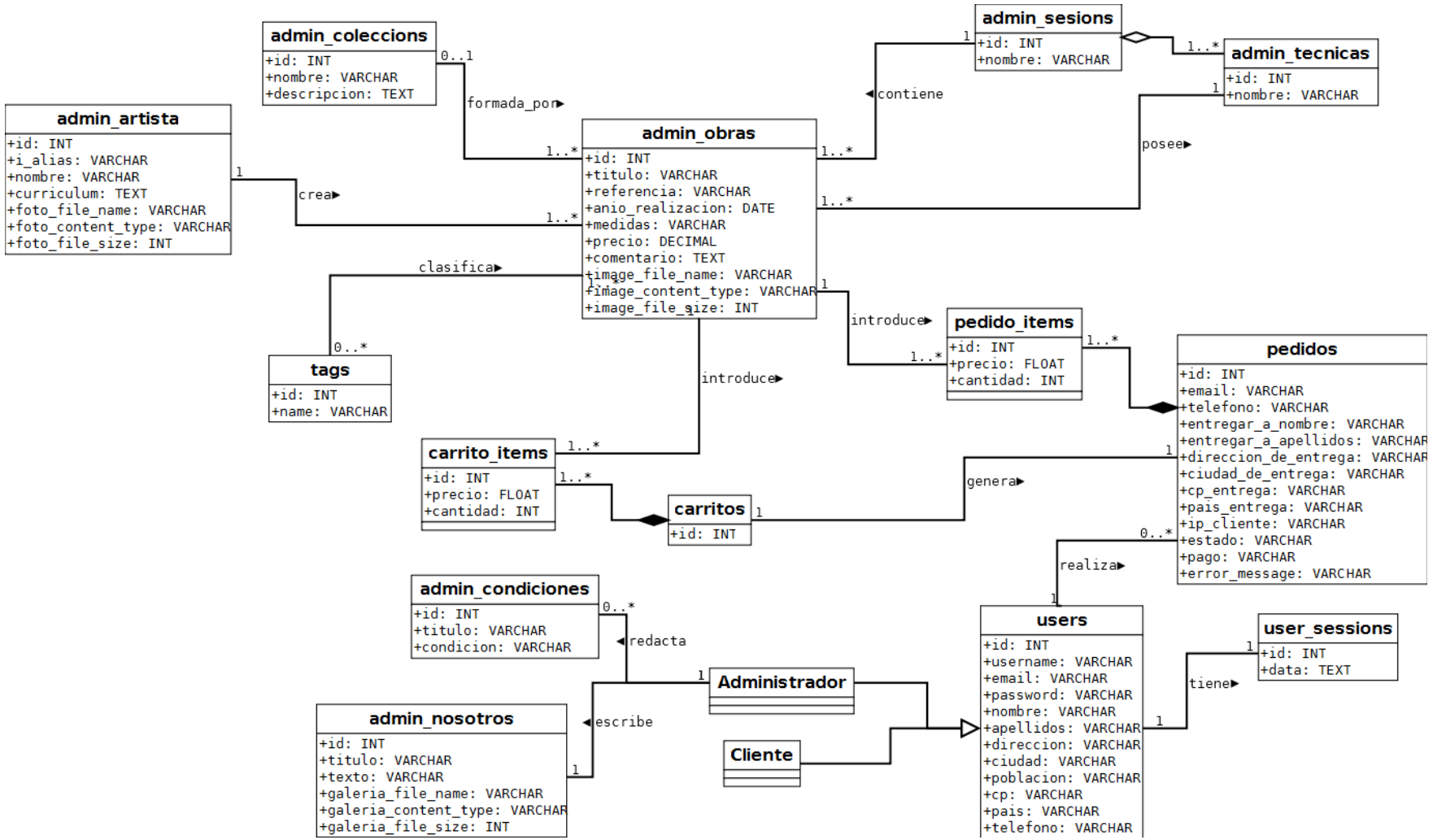


Figura 3.63: Diagrama conceptual de clases UML

3.4. Requisitos no funcionales

Esta sección contiene los requisitos detallados del sistema.

3.4.1. Requisitos de interfaz externa

A continuación se van analizar cada uno de los requisitos relativos a las entradas y salidas del software.

3.4.1.1. Interfaces de usuarios

El sitio web deberá:

- tener una estructura clara, ordenando el contenido y las funciones de la aplicación en pestañas o apartados que abarquen todas las funcionalidades disponibles, según el perfil de seguridad del usuario conectado.
- posibilitar la visualización de cualquier tipo de contenido multimedia (texto, gráficos, videos, etc.) en consonancia con la imagen corporativa de la empresa.

3.4.1.2. Interfaces con el hardware

Los usuarios deberán disponer de un ordenador que posea una tarjeta de red o una tarjeta de red inalámbrica y un punto de acceso que les permita una conexión a Internet para poder acceder a la aplicación.

3.4.1.3. Interfaces con el software

Los usuarios necesitarán la utilización de un navegador web para que la aplicación pueda ser visualizada, no siendo necesario el uso de uno específico debido a que la aplicación estará adaptada para poder ser visualizada en la mayoría de los navegadores disponibles.

3.4.1.4. Interfaces de comunicaciones

La comunicación entre el cliente y el servidor, consiste en una comunicación de petición y respuesta. Estas se efectuarán con el protocolo HTTP y serán enviadas del cliente/servidor o del servidor/cliente con el protocolo TCP/IP.

3.4.2. Requisitos de seguridad

Los datos de la aplicación solo podrán ser modificados por aquellas personas autorizadas para ello.

3.4.3. Requisitos de eficiencia

3.4.3.1. Requisitos de espacio

La aplicación se alojará en un servidor contratado en Internet por lo que se debe tener en cuenta el espacio contratado, ya que un catalogo de productos con sus respectivas fotos puede requerir un espacio en el disco de tamaño importante.

3.4.3.2. Requisitos de rendimiento

Las respuestas a las peticiones de los usuarios externos se deberá generar en un tiempo razonable.

3.4.4. Restricciones del diseño

El diseño de la aplicación debe estar orientado hacia la facilidad de uso y a una rápida localización de las opciones, para que pueda ser accesible a cualquier usuario.

3.4.5. Restricciones de base de datos

Será necesaria la utilización de una *base de datos* para poder almacenar toda la información necesaria para el correcto funcionamiento de la aplicación. He optado por almacenar la misma en *MySql*.

3.4.6. Atributos de sistema de software

Fiabilidad. El sistema debe ser fiable en cuanto a los datos que ofrece a los usuarios clientes/visitantes como a la información de configuración que muestra al usuario administrador.

Integridad. Para salvaguardar la integridad de los datos, he optado por almacenar los mismos en una base de datos subyacente en un SGBD. Estos sistemas ofrecen mecanismos y herramientas de control de la integridad sin necesidad de una supervisión por parte del usuario.

Mantenimiento. El mantenimiento será llevado a cabo por el *Administrador del Sistema*, a quien se le facilita un módulo de administración para realizar todas las tareas necesarias.

3.5. Reglas de negocio

Como ya mencioné en la sección 3.1, los perfiles de usuario de la aplicación serán tres: *administrador, usuario registrado (cliente) y usuario visitante*.

- El *administrador* tendrá acceso a todas las funcionalidades del módulo de administración, pudiendo desempeñar además el papel de usuario registrado normal.
- El usuario registrado tendrá acceso a un menú de operaciones que no incluya labores de administración, pudiendo realizar compras.
- El *usuario visitante* tendrá acceso a un menú de operaciones que no incluya labores de administración, pudiendo visitar las obras contenidas en el catálogo sin poder realizar su compra.

3.6. Estudio de alternativas tecnológicas

Para la elección del lenguaje he realizado un *análisis de la relevancia de los principales lenguajes de programación* y una vez seleccionado, he efectuado una comparativa con los distintos framework que lo utilizan.

3.6.1. Lenguajes

Determinar el lenguaje de programación, que se ajusta mejor a las necesidades específicas del proyecto, es un aspecto importante cuando se va a realizar una aplicación Web.

Hoy en día podemos encontrarnos una gran variedad de lenguajes de programación y determinar cuál es el más adecuado puede resultar complicado. Por esta razón, para realizar la selección, sería bueno tener en cuenta una serie de factores.

Para comenzar descartaré todos los lenguajes que su desarrollo requieran el pago de cualquier licencia.

Tendremos en cuenta varios factores a la hora de aprender un nuevo lenguaje de programación:

- la eficiencia,
- la facilidad de escribir,
- la alta calidad y
- que sea bastante productivo.

Ya que actualmente existen muchos lenguajes de programación con sus ventajas y desventajas, voy a realizar un *análisis de la importancia de los principales lenguajes de programación* dentro de la industria del software y así, podré determinar cual es el que aporta mejores prestaciones. Para ello ejecutaré consultas a múltiples servicios que cuentan con un gran tráfico de usuarios.

3.6.1.1. Análisis

He evaluado para el estudio, a través de la web **Programming Language Popularity** [Welton, David N.], servicios como *Freshmeat* [Geeknet], *Ohloh* [Software], *Google Code* [Google], *Delicious* [AVOS], *Craigslist* [CraigsList], *Indeed* [Indeed] y *Yahoo Search* [Yahoo]. De los resultados obtenidos he realizado una valoración global y para finalizar he estudiado la tendencia de la programación.

Freshmeat, Ohloh y Google Code

Son sitios web de referencia dentro de la comunidad de desarrolladores *Open Source*.

Google Code y *Freshmeat* son servicios web donde se almacenan todo tipo de proyectos Open Source. Sus análisis podemos observarlos en las *Figuras 3.64* y *3.65*.

Ohloh es un directorio público de Open Source y personas que lo desarrollan y/o usan. Se centra en un análisis de los proyectos que tiene enlazado.

Gracias a los datos que obtiene de los repositorios oficiales de los proyectos puede realizar diferentes informes sobre el uso del lenguaje de programación, el sistema de control de versiones o incluso comparar varios proyectos. Podemos ver su análisis en la *Figura 3.66*.

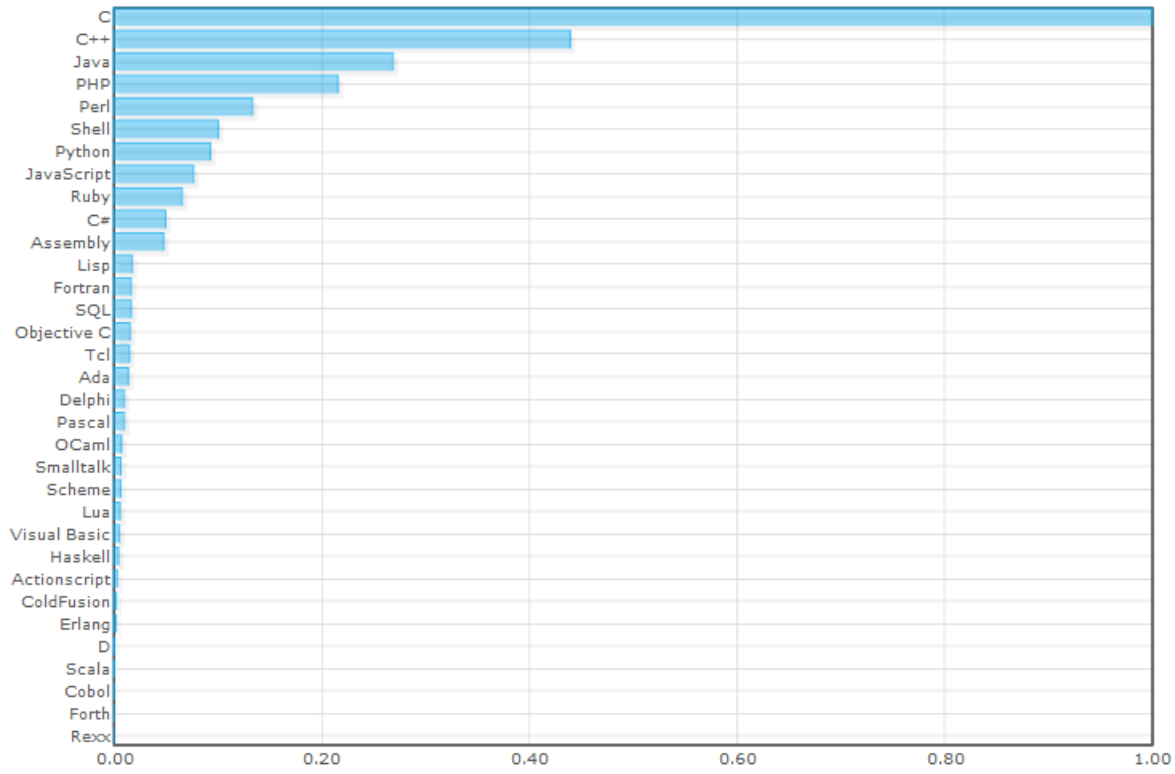


Figura 3.64: Índices de aceptación de lenguajes de programación - Google Code

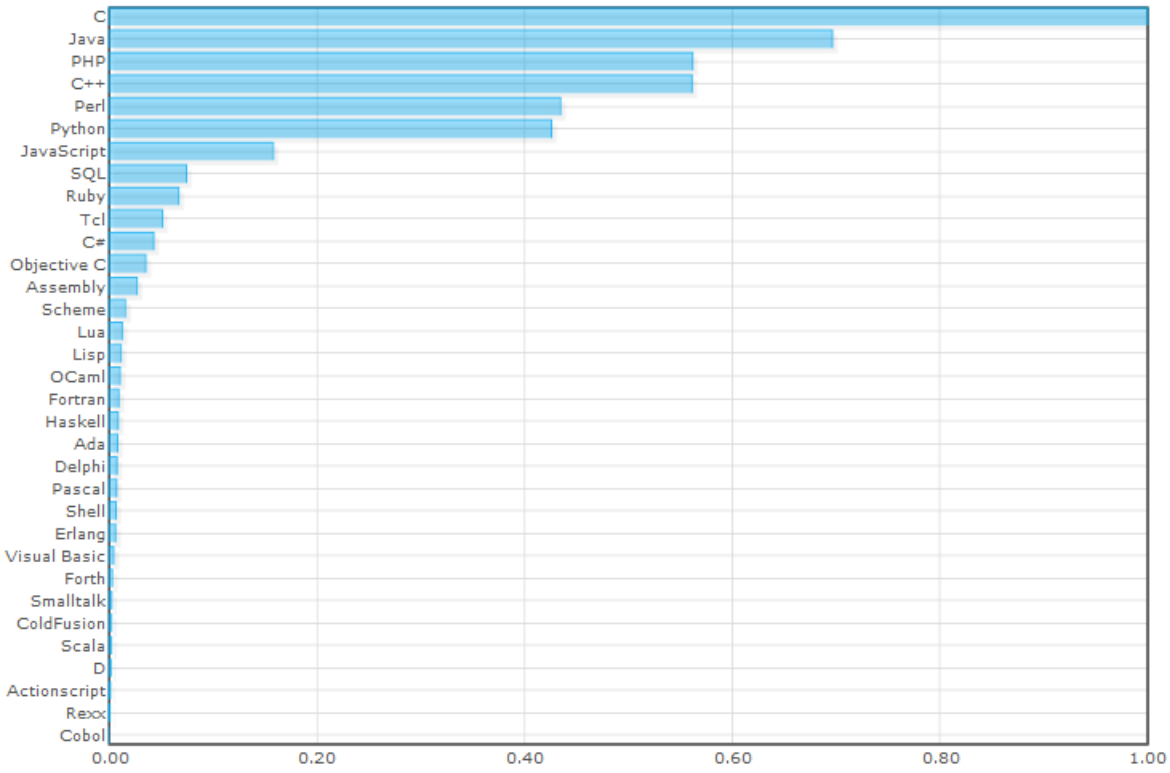


Figura 3.65: Índices de aceptación de lenguajes de programación - Freshmeat

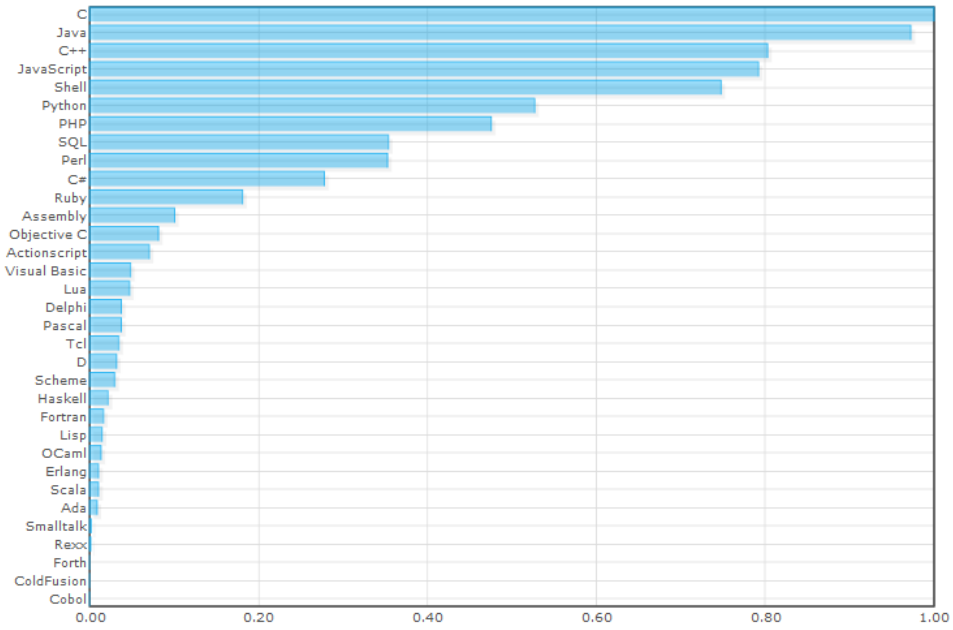


Figura 3.66: Índices de aceptación de lenguajes de programación - Ohloh

La elección del lenguaje vendrá determinada por las preferencias del programador y, aunque estas estadísticas no son significativas de las necesidades en la industria del software, si nos indican las preferencias de los programadores en el uso de un lenguaje.

Delicious

Es un servicio de gestión de marcadores sociales en web. Permite agregar los marcadores y categorizarlos con un sistema de etiquetado. *Delicious* permite almacenar sitios webs y compartir los marcadores con el resto de sus usuarios.

Para estudiar los intereses de una comunidad de millones de usuarios, he realizado un análisis del número de referencias a los marcadores referidos a los lenguajes de programación, obteniéndose los resultados en la *Figura 3.67*.

Craigslist, Indeed

Para obtener las prestaciones y el grado de productividad de un lenguaje, hay que estudiar el nivel de aceptación que tiene en la industria del software. Esto queda reflejado en el número de ofertas de empleo que demandan programadores con conocimientos de un lenguaje específico. Para ello he evaluado dos sitios web.

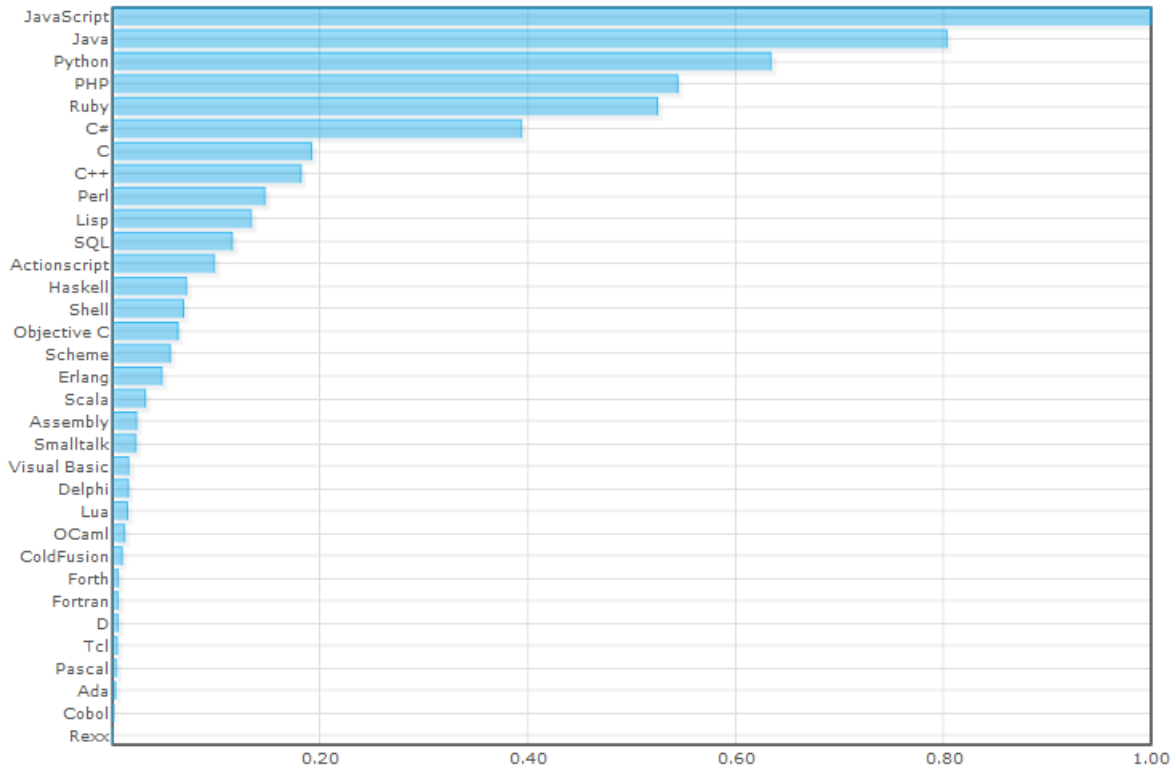


Figura 3.67: Índices de aceptación de lenguajes de programación - Delicious

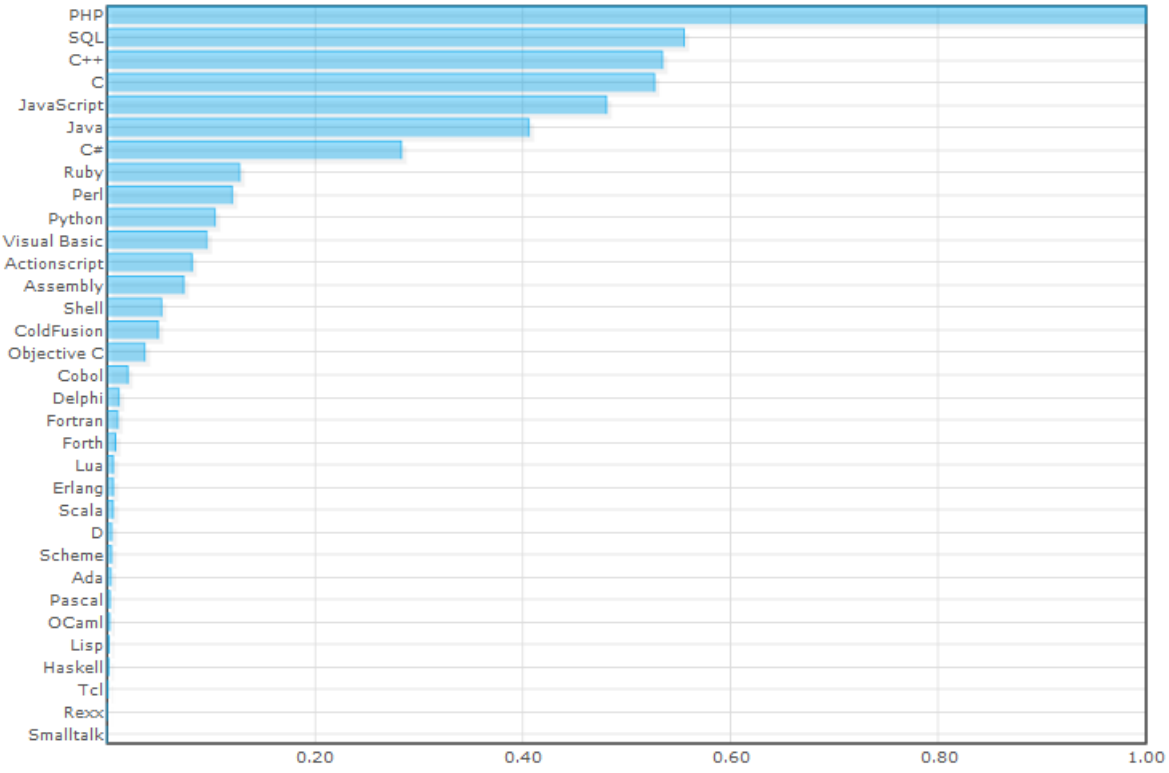


Figura 3.68: Índices de aceptación de lenguajes de programación - Craigslist

En la figura 3.68 podemos observar el número de ofertas de trabajo actuales obtenidas del sitio web *Craigslist*.

En el sitio web *Indeed.com* he realizado un análisis de las tendencias de la demanda de programadores con conocimientos específicos de un lenguaje de programación.

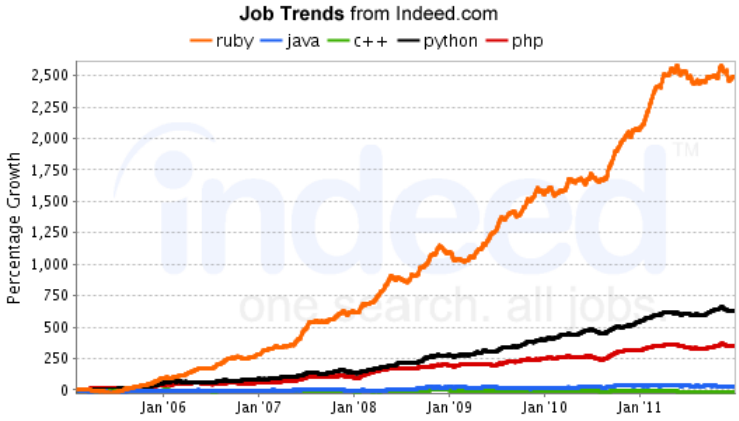


Figura 3.69: Índices de aceptación de lenguajes de programación - Indeed

La primera estadística que podemos observar en la *Figura 3.69*, presenta la tendencia relativa de crecimiento, donde se puede comprobar cómo en los últimos años la demanda se ha mantenido estable, con la distinción de un crecimiento considerable de la necesidad de programadores Ruby.

Este crecimiento de Ruby, se puede considerar normal ya que es un lenguaje joven que está teniendo una gran aceptación.

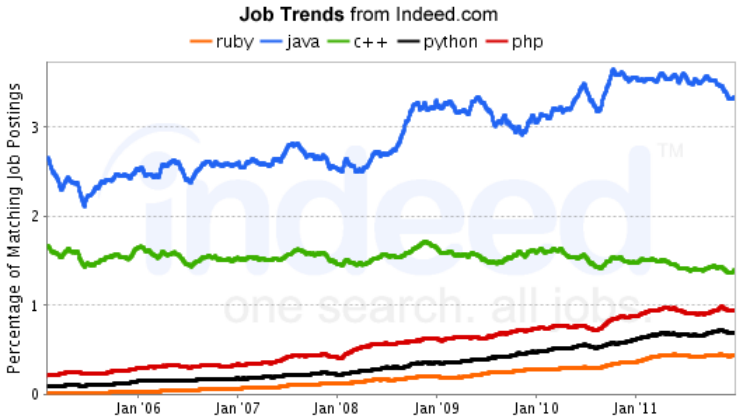


Figura 3.70: Índices de aceptación de lenguajes de programación - Indeed

En la segunda gráfica que se muestra en la *Figura 3.70*, se puede ver la tendencia de la demanda en valores absolutos de los diferentes lenguajes. Se puede comprobar que *Java* es

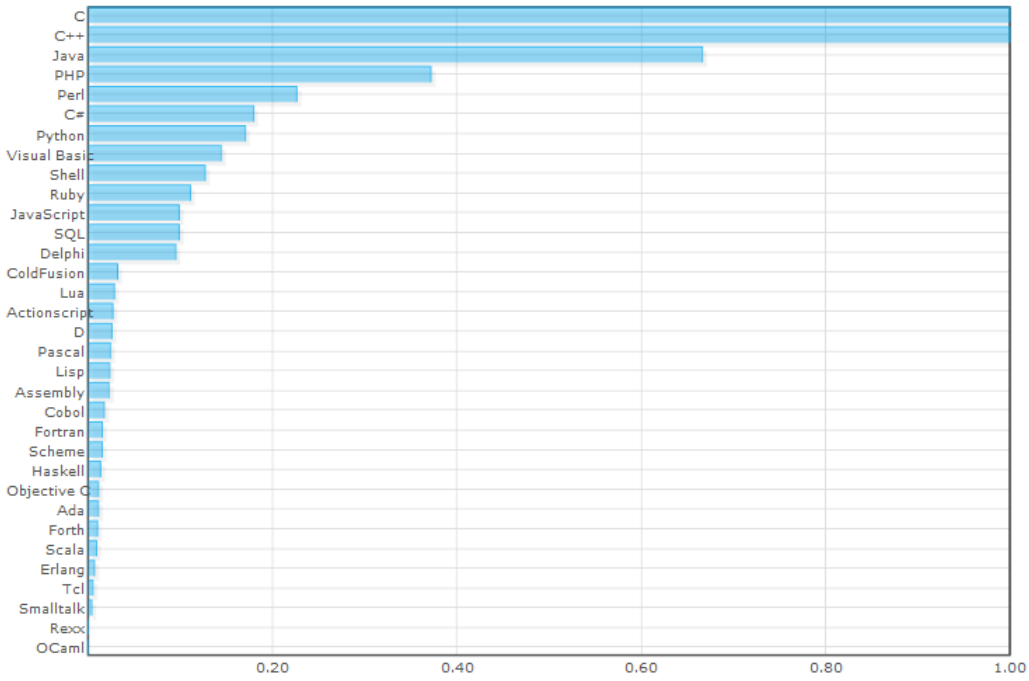


Figura 3.71: Índices de aceptación de lenguajes de programación - Yahoo search

el lenguaje más demandado en el mercado en los últimos años. Esto, junto con los datos obtenidos anteriormente del estudio, nos indican que *Java* es el lenguaje de desarrollo por el que se decanta la comunidad de programadores.

Yahoo Search

Es un motor de búsqueda de páginas web. Si realizamos una búsqueda genérica por *"language programming"* se obtiene una clasificación con el número de referencias que hay de cada lenguaje en Internet. Podemos verlo en la *Figura 3.71*.

Conclusión

Tomando como referencia la tendencia de ofertas de trabajo absolutas (*Figura 3.68* de la *página 88*) y agrupando las estadísticas presentadas por las diferentes Webs analizadas, se obtiene la conclusión de que los lenguajes de programación más populares son **Java, PHP** [Gómez López, Rubén] y **C/C++**. Ello podemos observarlo en la *Figura 3.72*.

C/C++ se pueden considerar lenguajes de programación de más bajo nivel que *Java*, lo que aporta gran versatilidad y rendimiento a sus aplicaciones. La dependencia que a veces tienen sus librerías del sistema operativo, la escasez de frameworks de desarrollo web que existen y su complejo sistema de conexión a bases de datos, descartan estos lenguajes a la hora de desarrollar aplicaciones Web.

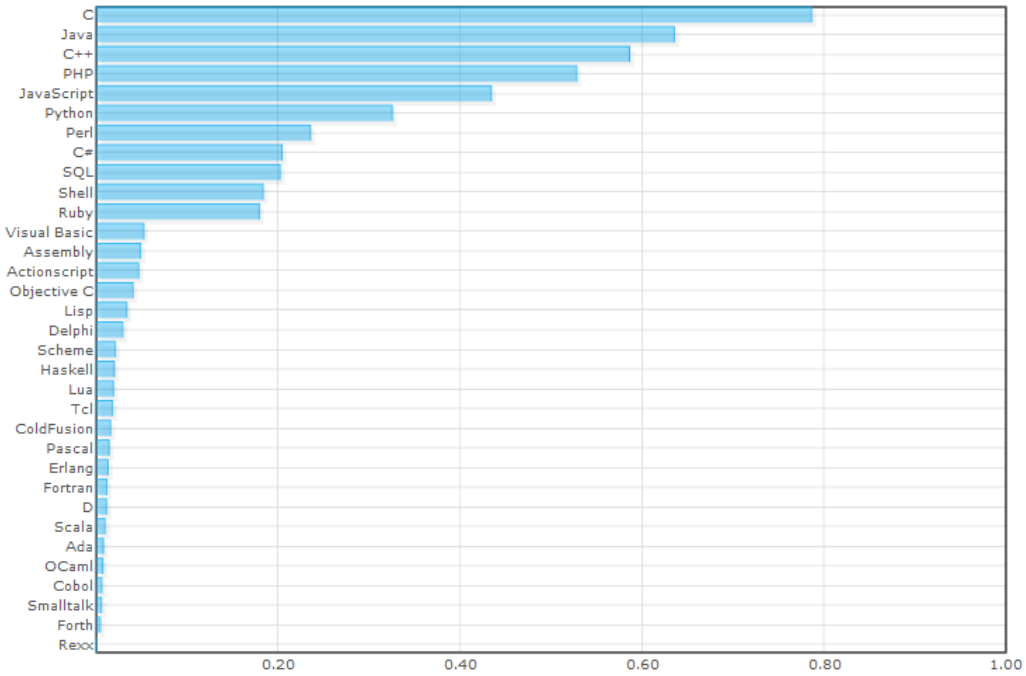


Figura 3.72: Índices de aceptación de lenguajes de programación - Global

Java tiene un manejo muy bueno de conexiones a base de datos, la portabilidad entre sistemas operativos es muy buena ya que ejecuta el código Java en su propia maquina virtual, no haciéndola dependiente de librerías propias de un sistema operativo y tiene una serie de frameworks de desarrollo de aplicaciones Web que hacen que este lenguaje sea altamente productivo a la hora de desarrollar aplicaciones Web.

PHP es uno de los lenguajes mas populares en el desarrollo de aplicaciones web. Quizás una de las ventajas frente a *Java* sea en cuestión de *rendimiento* ya que *PHP* es mucho menos pesado, lo que produce una sensación al usuario de rapidez y mayor usabilidad. El coste estimado en estos proyectos será menor, ya que la programación de un sistema *PHP* es mucho más directa con resultados inmediatos.

Siempre existen diferentes variables que nos influyen a la hora de tomar una decisión y que se deben sopesar para encontrar un equilibrio entre lo que la tecnología nos ofrece y lo que necesitamos nosotros de ella. Pero las variables más importantes a tener en cuenta serán *el propósito de la aplicación web y sus dimensiones.*

Veamos en la *Gráfica 3.73*, a modo estimativo, la comparación entre ambas tecnologías.

3.6.1.2. Ruby como lenguaje para mi proyecto

Uno de mis objetivos, en este proyecto, es el poder adquirir nuevos conocimientos en lenguajes de programación orientados a la web. Dado que durante la carrera he tenido la oportunidad de aprender algunos de los lenguajes anteriormente mencionados como más

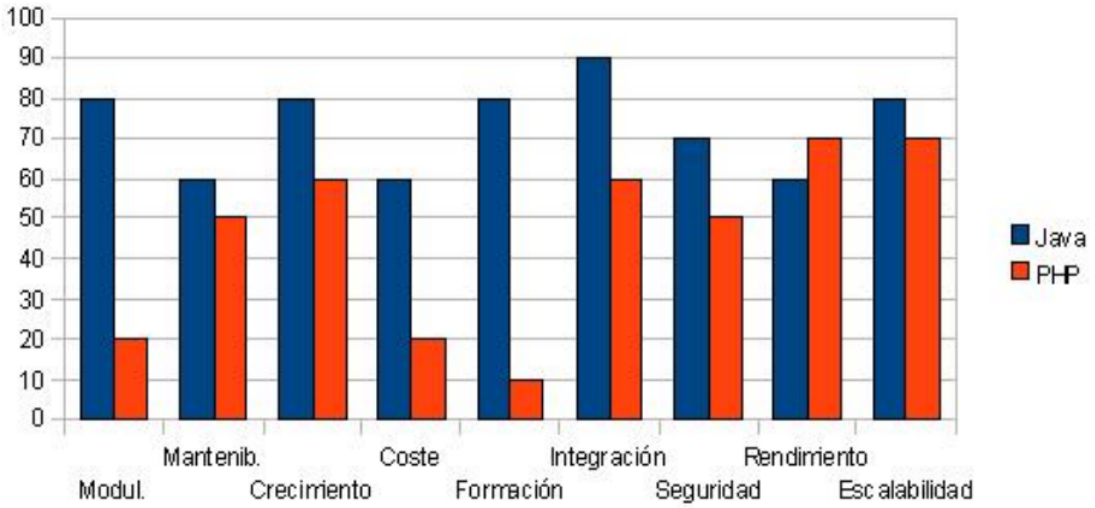


Figura 3.73: Comparativa PHP - Java

populares, C/C++ (orientado a objetos), y que tengo algunas nociones sobre *Java* de un curso que realicé hace ya algunos años, me he decidido por aprender un nuevo lenguaje *Ruby* (ver *anexo D*), ya que parece ser bastante interesante y presenta una nueva forma de programar diferente a las estudiadas durante la carrera.

3.6.2. Entorno de desarrollo Web

Se puede decir que un *Entorno de Trabajo Web*, en inglés *«Framework»*, es un conjunto de aplicaciones, utilidades de soporte y código que simplifican las tareas de los programadores, nos facilitan una infraestructura que nos ahorra muchas horas delante del ordenador.

Estos pueden ser una herramienta muy útil, cuando tenemos mucho trabajo, para ahorrarnos tiempo y ayudarnos a ser más eficientes en nuestro día a día, aunque por supuesto, podemos decidir desarrollar todo el código.

Estos persiguen los siguientes objetivos:

- acelerar el proceso de desarrollo,
- reutilizar código ya existente y
- promover buenas prácticas de desarrollo, como el uso de patrones.

Utilizando un framework para el desarrollo del proyecto conseguimos algunas ventajas como:

- El programador **no necesita plantearse** una estructura global de la aplicación, sino que el framework le proporciona un esqueleto que hay que "rellenar".

- **Mayor velocidad** a la hora de realizar los desarrollos, ya que en muchas ocasiones ofrecen funcionalidades definidas que nos facilitan el trabajo.

- Facilita la **colaboración**. Cualquiera que haya tenido que "pelearse" con el código fuente de otro programador o incluso, con el suyo mismo pasado algún tiempo, sabrá lo difícil que es entenderlo y modificarlo; por tanto, todo lo que sea definir y estandarizar va a ahorrar tiempo y trabajo a los desarrollos colaborativos.

- Es más fácil encontrar **herramientas** (utilidades, librerías) adaptadas al framework concreto para facilitar el desarrollo.

- **Código optimizado**, ya que por norma se encuentra un grupo de personas trabajando sobre el mismo que intentan que el código sea lo más limpio y eficiente posible.

- **Reducción de costos**, ya que al mejorar la velocidad de desarrollo y obtener un código mucho mejor, los costes de desarrollo también se reducen.

La utilización de un framework en el desarrollo de una aplicación implica un cierto coste inicial de aprendizaje, aunque a largo plazo es probable que **facilite** tanto el **desarrollo** como el **mantenimiento**.

Actualmente se pueden encontrar una gran variedad de *Entornos de Desarrollo Web* que están basados en *Ruby*, y determinar cuál es el más adecuado para el presente proyecto puede resultar complicado. Por ello se debe tener en cuenta una serie de factores para garantizar una correcta valoración de las prestaciones ofrecidas por cada uno ellos.

Tal y como ya mencioné anteriormente en la *Sección 3.6.1.2 "Ruby como lenguaje para mi proyecto"* de la *página 91*, uno de los objetivos del presente proyecto es el *adquirir y reforzar nuevos conocimientos acerca de la programación de aplicaciones Web*. Es por lo debo tener en cuenta en la elección que, aparte de que facilite las tareas repetitivas, me ayude a programar siguiendo las *"buenas prácticas"* de programación.

La filosofía de diseño *MVC (Modelo-Vista-Controlador)* es una de estas buenas practicas. Este patrón de diseño (descrito en la *Sección 4.1.2 "Arquitectura del sistema"* de la *página 104*) se emplea para separar, en la interfaz de usuario, la lógica de negocio de la representación de la información. Al seguir este patrón se incrementa la *flexibilidad* y la *reutilización* al facilitar la modificación de la lógica de negocio o de la interfaz de usuario, sin que una afecte a la otra.

Dentro de este patrón se debe diferenciar entre dos arquitecturas:

- **Arquitectura Push-based**. Se conoce también como *server push*, caracterizándose en que la petición para que se sirva una determinada información se produce en el servidor, implicando que este se encarga de inyectar los contenidos hacia el cliente, sin que se produzca la petición previa.

- **Arquitectura Pull-based**. Las peticiones de información se originan en el lado del cliente y el servidor solo se debe encargar de enviar la respuesta a estas peticiones.

A continuación podemos ver una comparativa de los entornos de desarrollo web de Ruby en la siguiente tabla:

Framework	Ajax	MVC	MVC Push/Pull	Internacionalización y localización	ORM	Testing	Seguridad	Uso de plantillas	Uso de caché	Validación de formularios
Camping	No	Si	Push	No	Patron Active record	via Mosquito	No	Si	No	No
Nitro	jQuery	Si	Push	Si	Og	RSpec	Si	Si	Si	Si
Ruby on Rails	Prototype, script.aculo.us	ActiveRecord Action Pack	Push	Localization Plug-in	ActiveRecord	Unit Tests, Functional Tests and Integration Tests	Plug-in	Si	Si	Si
Sinatra	No	Si	Push	No	ORM-independent	Rack::test	No	Si	A traves del middleware de Rack	No

Tabla 3.3: Comparativa de entornos de desarrollo Web - Ruby

En la *sección 3.7 Análisis GAP* trataré sobre las características mostradas en la tabla anterior.

3.6.2.1. Rails como framework para mi proyecto

Como entorno de desarrollo web para mi proyecto he seleccionado **Ruby on Rails** (ver *anexo G*), también conocido como **RoR**.

Los motivos de esta decisión son:

- En la actualidad *Ruby on Rails* es el Framework más completo de todos los analizados anteriormente, lo que lo convierte en una herramienta de grandes prestaciones, tanto en productividad, flexibilidad y facilidad de mantenimiento.

- *RoR*, dentro de la comunidad de programadores de *Ruby*, es el Framework más popular aportando un apoyo absoluto por parte de desarrolladores, teniendo un efecto en un crecimiento exponencial de la comunidad Rails (Foros, Lista de correos, Screencast, Blogs, Libros, Documentación online, etc. . .)

- Si estudiamos la demanda de la industria del software (*Figura 3.74*) podemos ver como el sector apuesta fuerte por este entorno. Estas circunstancias eligen a *Ruby on Rails* como un entorno de desarrollo con grandes perspectivas de futuro dentro del panorama de aplicaciones Web.

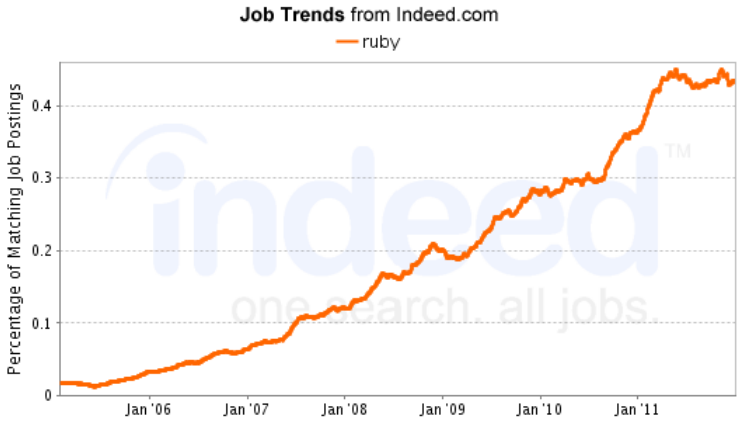


Figura 3.74: Tendencia del crecimiento de la demanda de Rails - Indeed.com

3.6.2.2. Otros frameworks

También he querido analizar algunos de los frameworks más utilizados en comercio electrónico *B2C*. Aunque es difícil compararlos con *Rails* ya que se basan en un lenguaje diferente: *PHP*.

Cabe destacar **Oscommerce**, **Magento** y **Prestashop** como las mejores aplicaciones opensource, libres y gratuitas.

Oscommerce

Este gestor de tiendas apareció en Marzo del año 2000 y posiblemente haya sido el mejor gestor para tiendas online, ahora bien, a día de hoy, se ha quedado atrás y más viendo lo fuerte que vienen sus competidores. Hay que destacar su estabilidad y potencia, en cambio tiene en contra su diseño con tablas, que en los años en los que nos encontramos no podemos permitir seguir usando maquetación sin CSS.

Actualmente está la versión 3 que apareció corrigiendo grandes problemas que tenía con los logins, dispone de un código obsoleto que puede traer de cabeza a más de uno debido al gran número de agujeros que presenta..., también separaron lo que es código del diseño, pero aún con estas mejoras sigue existiendo una huida masiva de este sistema hacia otras plataformas como son *Prestashop* y *Magento*.

Dado que *Oscommerce* está desactualizado y descontinuado, no creo que sea una buena opción para desarrollar mi proyecto.

A destacar:

- Permite integrar varios idiomas.
- Gran comunidad de desarrolladores, en 2011 la mayor comunidad en español desaparece.
- Gran cantidad de módulos desarrollados por lo que se abaratan costes.
- Gestión de multitud módulos de pago.
- Gestión de envíos, ya sean por zonas, tramos de pesos, etc....
- Desarrollar en Oscommerce es más económico, es más sencillo.
- Instalación muy sencilla.

En contra:

- La instalación inicial es muy básica, requieren de muchos módulos para comenzar a parecerse a una tienda.
- Apenas usa *CSS* por lo que todos los cambios de bloques hay que realizarlos manualmente.
- Proyecto estancado.
- Muy laboriosos, cualquier pequeña modificación requiere grandes conocimientos de *PHP*.
- No son accesibles (mapas del sitio, url amigables, meta-tags, títulos dinámicos,etc...) por parte de los buscadores si no implantamos un gran número de módulos.
- Con el cierre del foro en español de Oscomerce, desaparece la mayor ayuda que existía de este software en Español.
- Muchísimos bugs/errores en seguridad.

Prestashop

Este es uno de los gestores más nuevos, muy buena indexación, Ajax totalmente integrado, lo que le da un aspecto muy actual, y por supuesto funciona con *CSS*. Resulta muy fácil cambiarle el aspecto y personalizarlo.

El proyecto se creó en Francia, pero dispone de soporte en inglés, y por supuesto en español, aunque la comunidad española no es tan grande, sigue creciendo a buena marcha. La aplicación pesa muy poco y se instala con una facilidad sorprendente, en pocos minutos se tiene la tienda funcionando.

Llama la atención lo completo que es el panel de administración, la facilidad y lo intuitivo que son todos los paneles, los clientes lo agradecerán. Sobre todo cuando comprueben la facilidad con la que podrán gestionar toda la tienda, crear categorías, productos, fabricantes, controlar a los clientes, etc...

Es una aplicación modular por lo que nos permitirá ir añadiendo funcionalidades sin apenas tener que modificar el core de la aplicación.

Sin lugar a dudas en estos momentos parece ser el *rey* del comercio electrónico, empezó despacio pero disponer de una tienda realizada en Prestashop [Global] es garantizarse un buen desarrollo.

A destacar:

- Permite integrar varios idiomas.
- Coste final de proyecto profesional económico.
- Gestión de multitud módulos de pago. Es muy sencillo incorporar nuevos módulos para ampliar la características de la tienda.
- Grupos de clientes integrado.
- Fácil instalación con la mayoría de opciones.
- La herramienta atributos, personalizable y sencilla de usar.
- Permite definir productos físicos o virtuales (descargas).
- Muy fácil de usar.
- Bajo consumo de CPU.
- Permite introducir códigos de barras.
- Panel de administración muy intuitivo y sencillo.
- Es muy rápido.
- Prestastore: Tienda de módulos ya desarrollados.
- Prestashop 1.5 dispondrá de multitienda.
- La comunidad en español está creciendo muy rápido.
- Prestashop 1.5 dispondrá de versión para móviles tanto para Iphone como Android.

En contra:

- Soporte mayormente en Francés o Inglés.
- Sistema de atributos a mejorar.
- Escasos Módulos y themes.

Magento

Nació en el año 2007 siendo un proyecto relativamente joven, pero pese a este factor se ha ganado el respeto de muchos programadores, siendo hoy en día uno de los sistemas más utilizados.

Una de las cosas que no gusta de este gestor es su instalación, demasiados problemas para instalar el script, igual de complicado es su panel de control, si se tiene experiencia en páginas web, se consigue hacerse a él, pero si el cliente, que al final va a ser la persona que va a trabajar día a día en ella, no tienes conocimientos informáticos puede tener bastantes problemas para poder gestionarla.

Si se es un usuario novel no se recomienda *Magento*, en cambio si ya se tiene conocimientos de programación, se puede realizar prácticamente cualquier cosa.

Es recomendable para grandes empresas o grandes proyectos, ya que es donde realmente se ve su potencial, eso sí es muy importante saber modificar *Magento* porque la inexperiencia puede ser fatal, ya que el rendimiento caerá en picado con una mala programación.

La gestión de pedidos la realiza realmente bien.

Los principales problemas que presenta *Magento* son:

- La lentitud, difícilmente funcionara en un hosting compartido, necesita mínimamente un VPS o un servidor dedicado para poder correr la tienda en condiciones, aparte de tener instalado aplicaciones de rendimiento.
- Para tocar el corazón de *Magento* es importante tener los suficientes conocimientos ya que cualquier error afectará seguro a la aplicación y su rendimiento.
- Tienen una versión gratuita que cada vez van limitando más y más, frente a la versión de pago que ronda los 10.000 euros al año.

Como podemos ver *Magento* [Global] es potente, es muy bueno, pero un desarrollo con esta aplicación es caro y se debe disponer de un servidor con suficientes recursos.

A destacar:

- Muy potente, se puede realizar casi todo.
- Permite multitiendas.
- Sistema de búsquedas en Ajax.
- Permite una personalización completa del sitio.
- El panel de administración es el más completo con respecto a las herramientas anteriores.
- Gestión de pedidos muy potente.

En contra:

- Comunidad pequeña, poco soporte, prácticamente casi todo en Inglés.
- Costes finales altos.
- Instalación y personalización complicada.
- Módulos desarrollados o themes escasos.
- Panel de administración complicado, más todavía para empresas sin conocimientos informáticos.
- Consume muchos recursos.
- El tamaño del archivo de instalación es muy grande.
- Características del servidor bastantes exigentes.
- Muchos módulos de pago.
- En caso de tener una tienda con mucho tráfico es posible necesitar dos hosting por lo que el gasto se puede disparar.
- La curva de aprendizaje es muy alta.

Comparativa

Finalmente del sitio web *Indeed.com* he realizado un pequeño análisis de las tendencias de la demanda de programadores con conocimientos específicos en un framework.

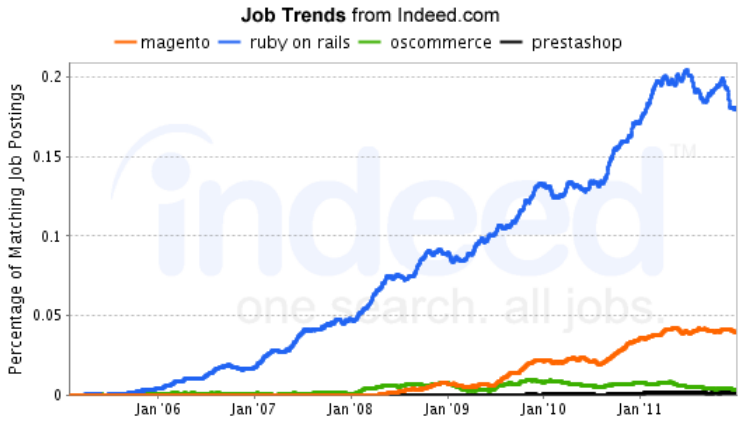


Figura 3.75: Índices de aceptación de frameworks - Indeed

La primera estadística, que podemos observar en la *Figura 3.75*, muestra la tendencia relativa de crecimiento. Donde se puede comprobar cómo en los últimos años la demanda de la necesidad de programadores que trabajen con *Rails* ha crecido considerablemente.

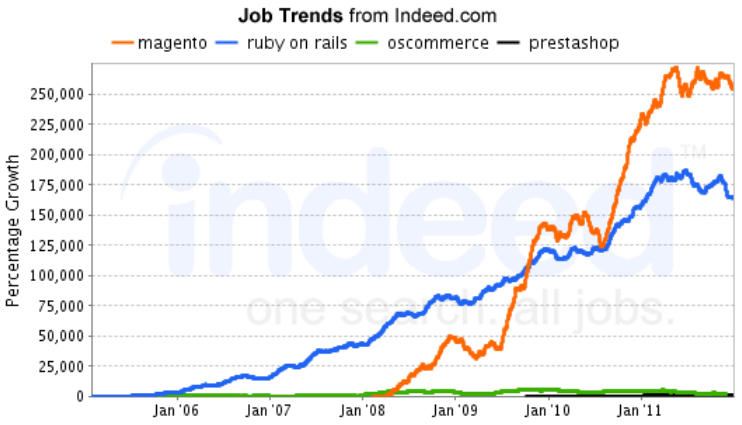


Figura 3.76: Índices de aceptación de lenguajes de programación - Indeed

En la segunda gráfica, que se muestra en la *Figura 3.76*, se puede ver la tendencia de la demanda en valores absolutos de los diferentes frameworks. Aquí se puede comprobar que *Magento* y *Ruby on Rails* son los entornos de desarrollo web más demandados por el mercado en los últimos años.

¿Porqué Ruby on Rails?

- *Ruby* es un lenguaje de propósito general muy flexible y moderno, cuyo propósito, en palabras de su creador *Yukuhiro Matsumoto*, es ayudar a los programadores a ser más productivos y que disfruten y se diviertan con la programación. Con esta concepción, el lenguaje se diseña para el programador, proveyendo de herramientas de alto nivel que hacen que sea muy cómodo y rápido trabajar con él.
- *Rails* implementa la mayoría de patrones de desarrollo que se utilizan en programación web, por lo que constituye una base muy sólida sobre la que arrancar nuevos proyectos, ahorra muchas horas de trabajo y nos ayuda a mantener nuestro código estructurado.
- *Código limpio*. Gracias a los principios de *convención sobre configuración* y *DRY - "No te repitas"*, el número de líneas de código a mantener en un proyecto Rails es mínimo si lo comparamos con un proyecto escrito en PHP o Java. Esto hace que sea mucho más flexible a cambios, y como todo programador sabe, menos código significa menos probabilidad de haber cometido un error.
- *Arquitectura del mundo real. Rails* fue originalmente extraído por su creador *David Heinemeier Hansson* de la aplicación *Basecamp*, una aplicación web de gestión de proyectos. Esto hace que toda su filosofía se haya creado sobre necesidades reales y le da un carácter muy práctico al framework.

3.7. Análisis GAP

El framework *"Ruby on Rails"* nos proporciona las siguientes características con las que puedo solventar los requisitos definidos para el proyecto:

- **Soporte para Ajax**. Mejoran la interactividad, velocidad y usabilidad de los sitios Web.
- **Patrón de diseño MVC**. Facilita la separación de la lógica de negocio de la interfaz de usuario.
- **Internacionalización y localización**. Soporte para poder adaptarse a diferentes idiomas y regiones sin necesidad de cambios de ingeniería, ni cambios en el código.
- **Interacción con bases de datos**. Incorporación de APIs para poder trabajar a alto nivel con diferentes tipos de bases de datos sin necesidad de modificar el código. Adicionalmente se pueden suministrar herramientas de mapeo entre objetos y estructuras relacionales (ORM), de gestión transaccional y de migración de bases de datos.
- **Testing**. Funcionalidades para realizar todo tipo de testing: test unitarios, funcionales y de integración.

- **Seguridad**. Funcionalidades para gestionar la autenticación y autorización a los usuarios en base a determinados criterios.
- **Uso de plantillas**. De esta forma se puede generar contenido de forma dinámica reduciendo drásticamente el número de páginas Web del sitio.
- **Uso de cache**. Se utiliza la cache para almacenar temporalmente documentos con el fin de reducir la carga del servidor y el tiempo de respuesta.
- **Validación de formularios**. Se suministran las herramientas necesarias para realizar la validación de la correcta entrada de datos en formularios Web.
- **Scaffolding**. Herramienta que nos permite crear un esqueleto funcional, que genera una interfaz web que permite llevar a cabo las operaciones básicas CRUD sobre una tabla especificada de la BD.
- **Tareas Rake predefinidas**. Con esta herramienta se pueden generar o automatizar el código.
- **WebServer**. Servidor para el desarrollo y las pruebas: WebBrick.

En la figura 3.4 vemos resumidas las características anteriores.

Ajax	MVC	Internacionalización y localización	ORM	Testing	Seguridad
Prototype, script.aculo.us	ActiveRecord Action Pack	Localization Plug-in	ActiveRecord	Test Unitarios, funcionales, integración	Plug-in

Uso de plantillas	Uso de caché	Validación formularios	Scaffolding	Rake	WebServer
Si	Si	Si	Si	Predefinidos	WebBrick

Tabla 3.4: Características de Ruby on Rails

Capítulo 4

Diseño del Sistema

La *fase de diseño* tiene como objetivo determinar cómo va ser construido el sistema a partir de los requisitos y el modelo obtenidos durante la especificación, aplicando patrones de diseño.

4.1. Diseño de la arquitectura

En esta sección se define la arquitectura general del sistema de información.

4.1.1. Arquitectura física

Los componentes que compondrán la arquitectura física son los siguientes:

- **Navegador del Cliente:** Un Navegador estándar HTML capaz de soportar CSS, Javascript + Document Object Model y XML. Este servirá como dispositivo de interfaz de usuario. Toda la interacción entre los usuarios y el sistema se realiza a través del navegador.
- **Servidor Web:** El navegador del cliente accederá al sistema a través del servidor Web (*WEBrick*), el cual acepta las peticiones del cliente y ejecuta, si es el caso, los scripts del lado del servidor necesarios. El resultado, una página HTML formateada, será enviada al cliente.
- **Conexión HTTP:** Es el protocolo más común actualmente entre el cliente y el servidor.
- **Conexión SMTP:** Protocolo de red utilizado para el intercambio de mensajes de correo electrónico.

- **Página dinámica:** Son páginas Web generadas por un procesamiento en el lado del servidor.
- **Servidor de Aplicaciones:** Es el principal motor para ejecutar la lógica del negocio del lado del servidor. En nuestro caso *WEBrick*, incluido con Ruby.
- **Servidor de Base de datos:** MySQL. Es la parte del sistema que mantiene el estado actual del negocio.

4.1.2. Arquitectura lógica

El haber seleccionado *Rails* para desarrollar mi aplicación implica la elección del *patrón de diseño de software* ***MVC*** *(«Modelo, Vista y Controlador»)* como base para mi proyecto. Este es una guía para el diseño de la arquitectura de aplicaciones que ofrezcan una fuerte interactividad con los usuarios.

La forma de organizar la aplicación consiste en separar:

- un ***modelo*** que representa los *datos de la aplicación* y sus *reglas de negocio*,
- un ***conjunto de vistas*** que representa los *formularios de entrada y salida de información*,
- un ***conjunto de controladores*** que procesa las *peticiones de los usuarios* y *controla el flujo de ejecución del sistema.*

La importancia de este modelo viene dada por la *reutilización de código.*

Arquitectura MVC en Rails

La rapidez en el desarrollo de proyectos con *Rails* está fundamentada en la idea de construir la aplicación separando en *3 capas* todos los componentes de la aplicación (*Modelo, Vista y Controlador*).

- **Modelo** *(Datos)* es el *SGBD («Sistema de Gestión de la Base de Datos»)* y las funciones que contienen la *lógica de negocio.*

  En las aplicaciones web orientadas a objetos sobre bases de datos, el *modelo* consiste en las *clases* que representan a las tablas de la base de datos.

  En el caso de *RoR* estas clases están gestionadas por la biblioteca de mapeo objeto-relacional (ORM) *ActiveRecord*, integrada en *Rails.*

  De esta forma, lo único que tiene que hacer el programador es heredar de la *clase ActiveRecord::Base*, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

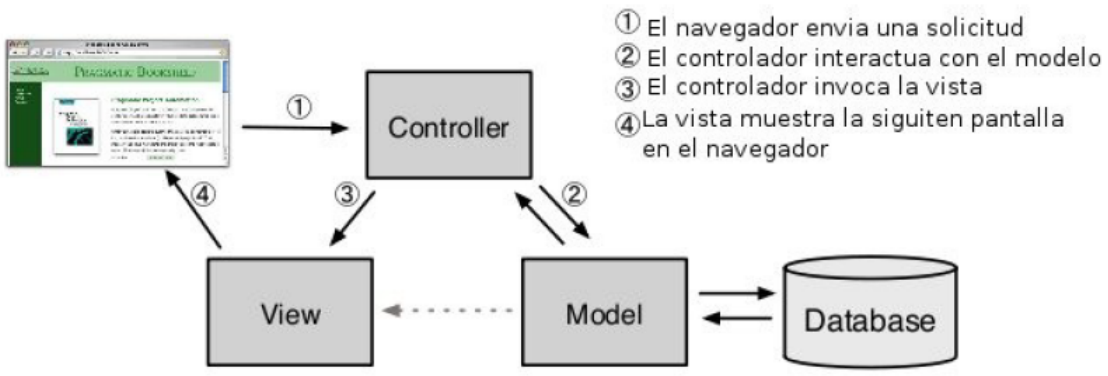


Figura 4.1: Arquitectura MVC

Un modelo puede tener varias vistas, cada una con su correspondiente controlador.

Es el código más reutilizable y más susceptible a cambios.

- **Vista** *(Interfaz de usuario)* es la *lógica de visualización*, o cómo se muestran los datos de las clases del Controlador. Con frecuencia en las aplicaciones web la *vista* consiste en una *cantidad mínima de código* incluido en *HTML*.

  Existen muchas maneras de gestionar las vistas. El método que se emplea en *Rails*, por defecto, es usar *Ruby Empotrado* ("archivos.html.erb"), que son básicamente fragmentos de código *HTML* con algo de código en *Ruby*, siguiendo una sintaxis similar a *JSP*. También pueden construirse vistas en *HTML* y *XML*.

  Es necesario escribir un pequeño fragmento de código en *HTML* para cada método del controlador que necesita mostrar información al usuario.

  Es la parte menos reutilizable y más fácil de modificar sin que se vea afectado nuestro núcleo.

- **Controlador** *(Lógica de negocio)* une la *vista* con el *modelo*, contiene toda la *lógica de programación*. Almacena las funciones que toman los valores de un formulario, delega consultas de base de datos al *modelo* y produce valores que invocarán a la *vista* adecuada.

  En el *MVC*, las clases del *Controlador* responden a la interacción del usuario e invocan a la *lógica de la aplicación*, que a su vez manipula los datos de las clases del *Modelo* y muestra los resultados usando las *Vistas*. En las aplicaciones web basadas en *MVC*, los métodos del controlador son invocados por el usuario usando el navegador web.

  La implementación del *Controlador* es manejada por el *ActionPack* de Rails, que contiene la *clase ApplicationController*. Una aplicación *Rails* simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, por lo general, en la forma *http://aplicacion/ejemplo/metodo*, que invoca a *EjemploController#método*, y presenta los datos usando el archivo de plantilla `\app\views\ejemplo\método.html.erb`, a no ser que el método redirija a algún otro lugar.

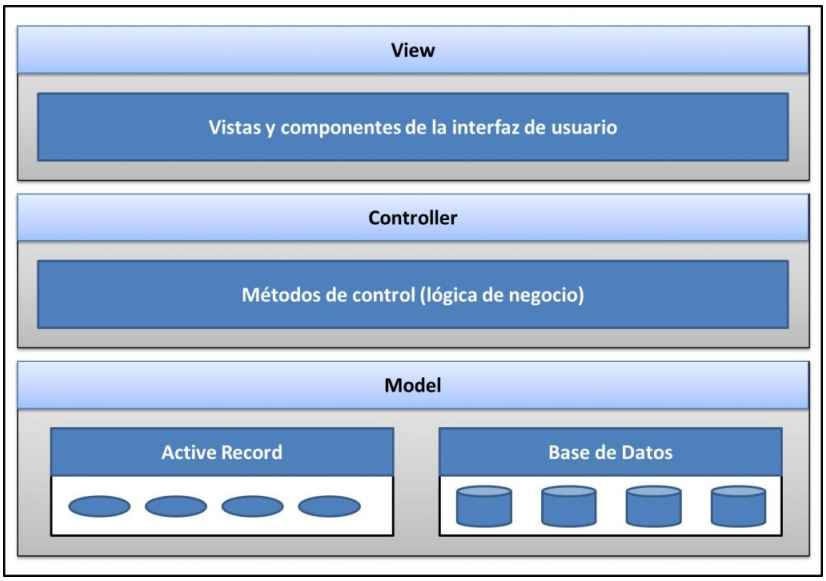


Figura 4.2: Representación gráfica en Rails del MVC

El diagrama de clase siguiente aclara cómo se implementa la arquitectura MVC en *Rails*.

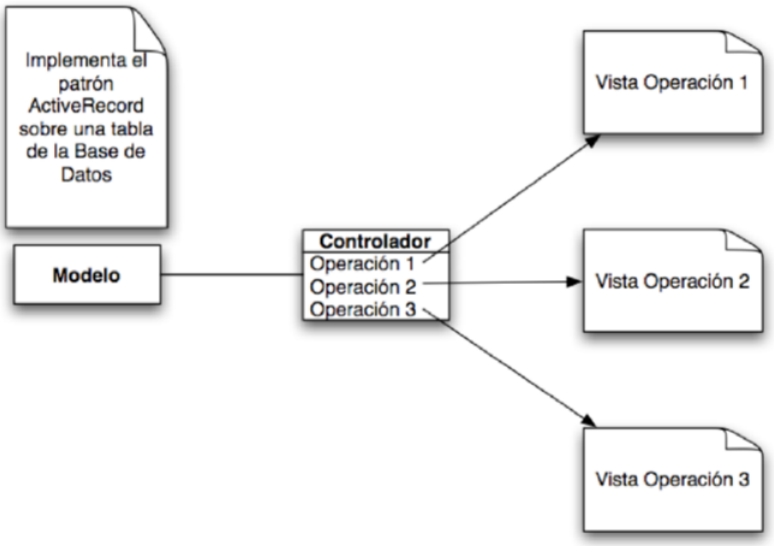


Figura 4.3: Implementación MVC en Rails

El *modelo* encapsula una tabla de la base de datos[1], el *controlador* contiene una serie de operaciones en los que se emplearán y modificarán los datos presentes en el modelo, y por cada una de estas operaciones, existirá una *vista*, desde la cual el usuario podrá configurar estas operaciones o recibir información sobre el estado de las mismas.

[1]Por convención, el nombre de la clase del modelo es el mismo de la tabla de la BBDD en singular.

Sobre esta convención se basa una de las prestaciones más útiles que ofrece *Rails*, conocida como ***Scaffold***. Consiste en un pequeño *script* que, a partir de una especificación de una tabla de una base de datos, puede construir rápidamente la mayor parte de la lógica y vistas necesarias para realizar las operaciones más frecuentes. (Ver *Sección 5.2 "Esqueleto para la aplicación"* de la *página 139*).

4.2. Diseño de la interfaz de usuario

Para el diseño de la interfaz gráfica de usuario he tenido en cuenta aspectos de usabilidad, ya que la aplicación desarrollada va a ser utilizada por personas que no están en la obligación de poseer conocimientos informáticos avanzados. Por ello la interacción con el usuario es simple e intuitiva.

El usuario recibe en todo momento una respuesta del programa en torno a operaciones consideradas como inválidas, indicándole cómo llevarlas a cabo de manera correcta mediante mensajes.

Al usar hojas de estilo CSS la apariencia de la aplicación puede ser fácilmente modificada para adaptarla a nuevas tendencias de diseño de forma sencilla y eficiente.

Cada una de las pantallas que forman la aplicación cuenta con una cabecera, diferente para el administrador, y un pie de página común.

Figura 4.4: Cabecera del administrador

Figura 4.5: Cabecera general

Figura 4.6: Pie de página

En la cabecera, se puede observar también un enlace para que el usuario se pueda registrar o acceder a su cuenta, así como otro para seleccionar el idioma: inglés o español. Estos aparecen en las diferentes pantallas de la aplicación.

Figura 4.7: Mockup - Link Registro/Acceso clientes

Si el usuario ha accedido a su cuenta, aparecerán los siguientes links en cada una de las pantallas.

Figura 4.8: Mockup - Acceso clientes

4.2.1. Mockups

En cada uno de los *sprints* de la aplicación he ido desarrollando una serie de mockups que me han ayudado a la hora de diseñar la interfaz de usuario. Éstos se han ido mostrando al cliente para su modificación y aceptación.

En el *anexo H "Relación de CU y Mockups"* podrá visualizar un listado que indica para cada *Caso de Uso* cual es el mockup que lo implementa.

A continuación se muestran algunos ejemplos de los mockup gráficos finales de la aplicación, mostrándose en la barra de direcciones la de cada una de las páginas. El resto de mockups podrá visualizarlos en la **carpeta "mockups"** contenida en el *CD adjunto*.

Ejemplos de mockups de la parte de administración. Como se puede ver en cada una de las pantallas, mediante el mensaje *"Bienvenido admin"*, solo el administrador tiene acceso a ellas.

Figura 4.9: Mockup - Panel de administración

En la pantalla anterior *(figura 4.9)*, el administrador tendrá acceso a los distintos gestores para poder administrar la web de forma sencilla. Veamos a continuación algunas un ejemplo de ellas para el *gestor de contenido*.

Visualizamos el mockup para la pantalla del gestor de contenidos *Obras*. Ésta tiene añadido un apartado para realizar las búsquedas, de esta forma el administrador podrá obtener un listado de las obras (figura 4.10) y también adquirir el listado realizando una búsqueda por cualquiera de los campos que se muestran.

En esta pantalla de la *figura 4.11*, el administrador verá toda la información referente al pedido de un cliente y podrá cerrar el pedido una vez haya sido enviado al cliente, cambiando de esta forma su estado a *cerrado*.

Ejemplos de mockups de la parte que puede visualizar cualquier usuario. Desde la pantalla de la *figura 4.12*, el usuario podrá tener acceso a las características de las diferentes obras así como añadirlas a la cesta de la compra.

La *figura 4.13* será utilizada para que el usuario pueda tener acceso, en caso de estar registrado, a su cuenta de usuario o para que pueda registrarse en la web.

La pantalla de la *figura 4.14* nos muestran la ficha técnica de las obras, podemos ver que el usuario también obtendrá algunas recomendaciones relacionadas con la obra que está visitando.

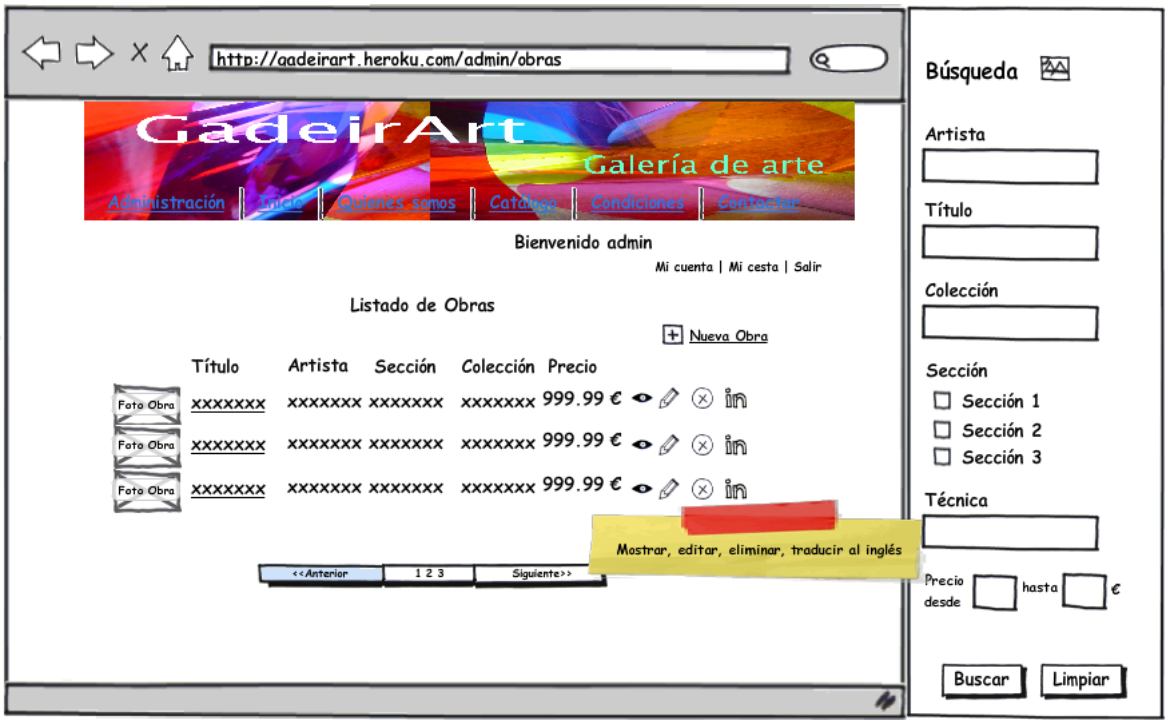


Figura 4.10: Mockup - Listado de obras

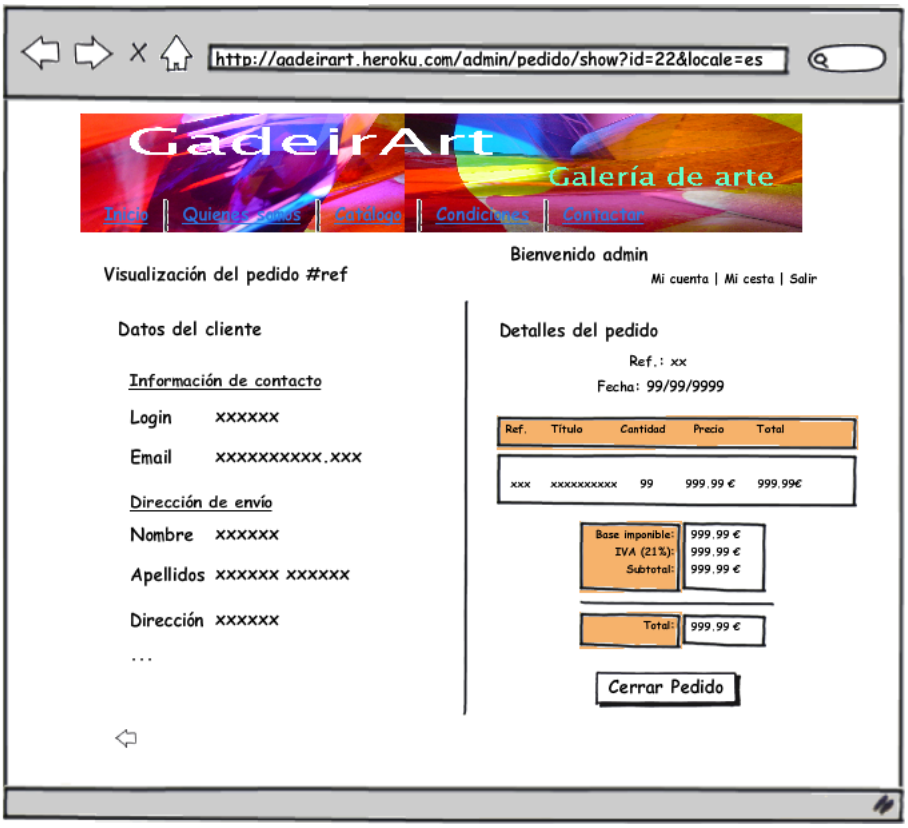


Figura 4.11: Mockup - Visualización del pedido

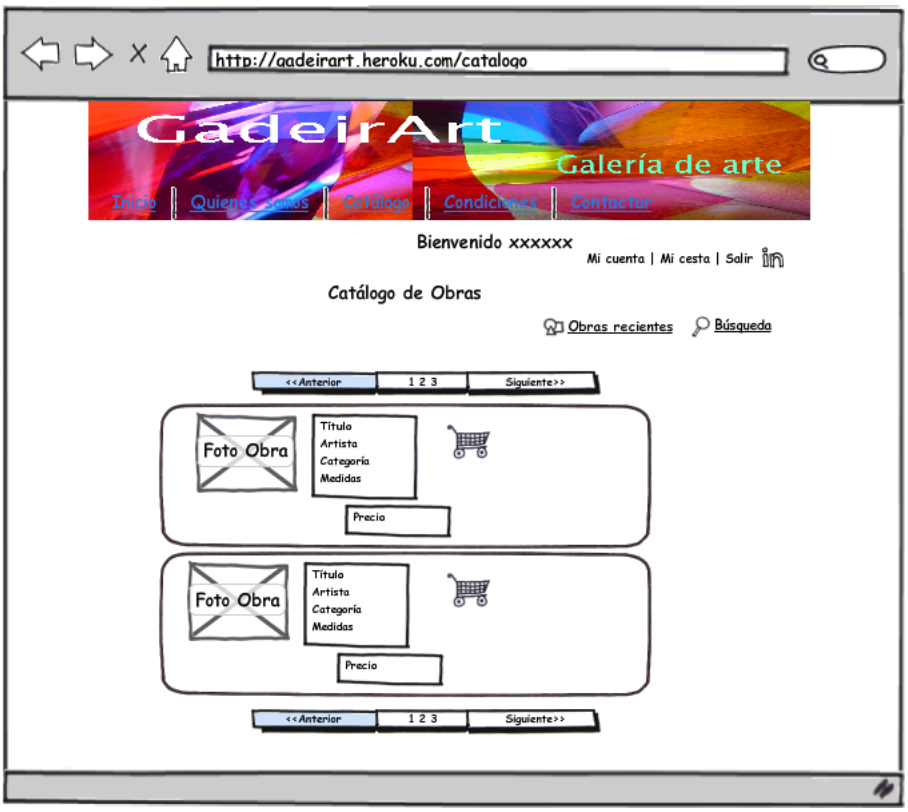


Figura 4.12: Mockup - Catálogo de obras

El siguiente ejemplo lo tenemos representado en la *figura 4.15* y es para la cesta de la compra, donde el usuario podrá visualizar las obras que desea adquirir así como el importe que le supondrá de manera desglosada. Desde aquí podrá realizar su pedido si está registrado, sino automáticamente le enviará a la pantalla para que realice el alta en la web.

También he realizado los mockups de los diferentes emails que se le envían al usuario y a la Galería. Un ejemplo de ellos lo tenemos representado en la *figura 4.16*, éste será recibido por el usuario cuando realice un pedido.

4.2.2. Diagramas de navegación

Mediante los siguientes *diagramas de navegación*, reflejaré la secuencia de pantallas a las que tienen acceso los diferentes roles de usuario y la conexión entre éstas.

Empezaré mostrando parte del diagrama de navegación para el *administrador*, concretamente el *menú de Administración* (figura 4.17), el cual he dividido para que se vea de forma más clara, pudiéndose visualizar a continuación el correspondiente para cada uno de sus gestores (figuras 4.18, 4.19, 4.20).

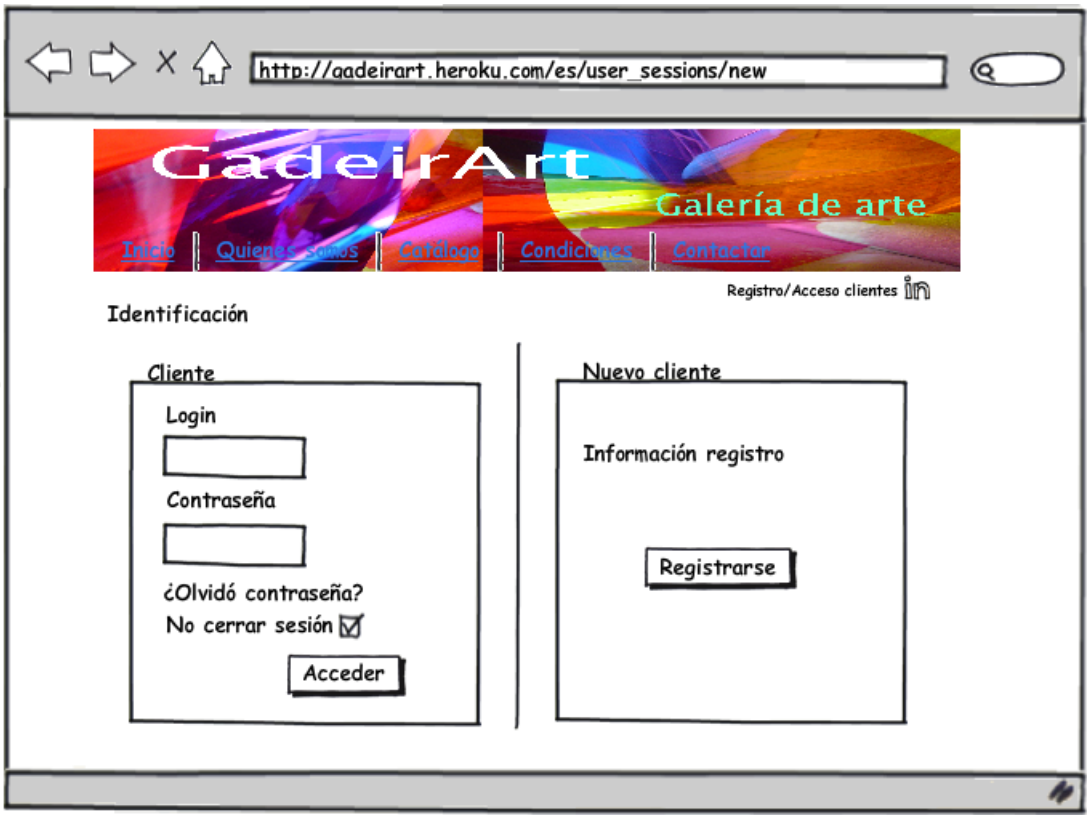


Figura 4.13: Mockup - Identificación

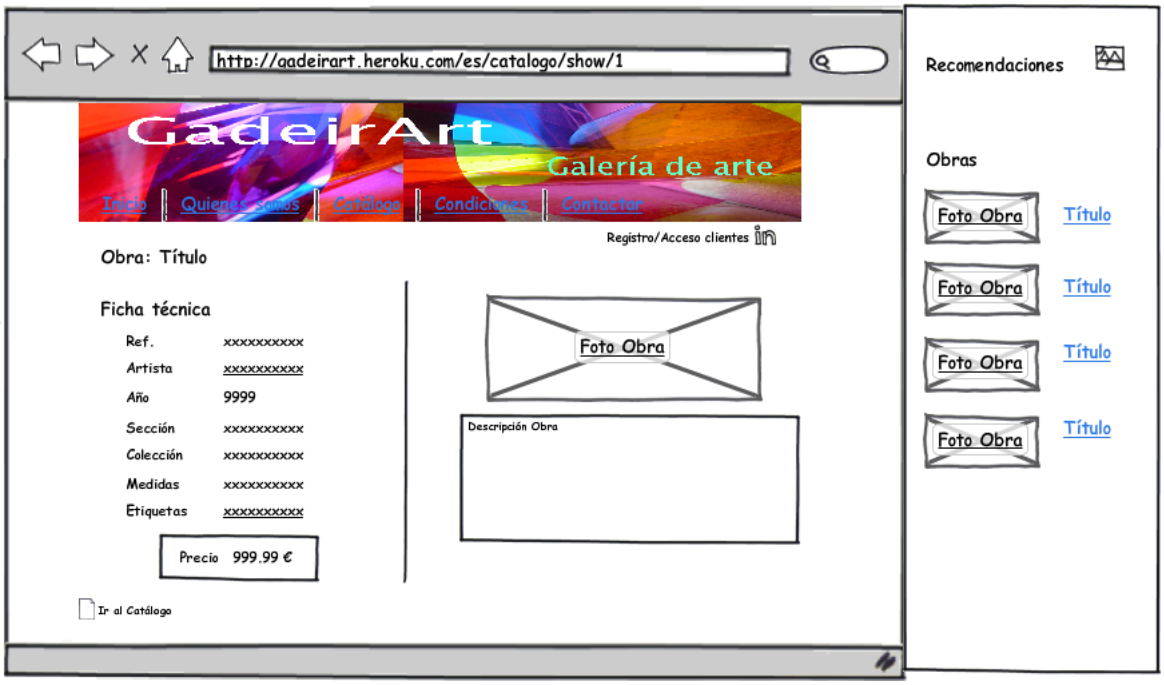


Figura 4.14: Mockup - Ficha de la obra

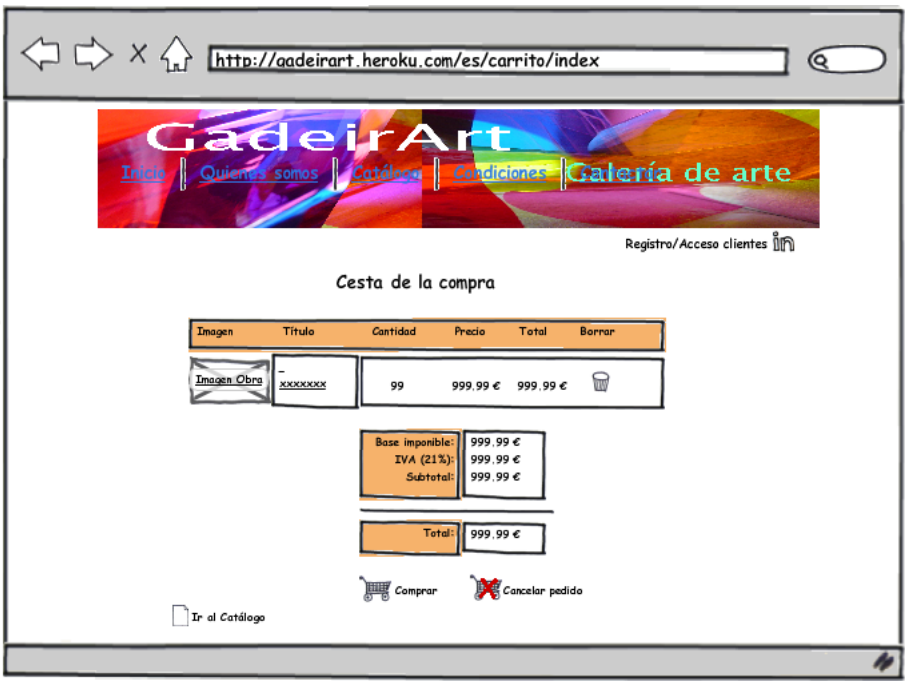
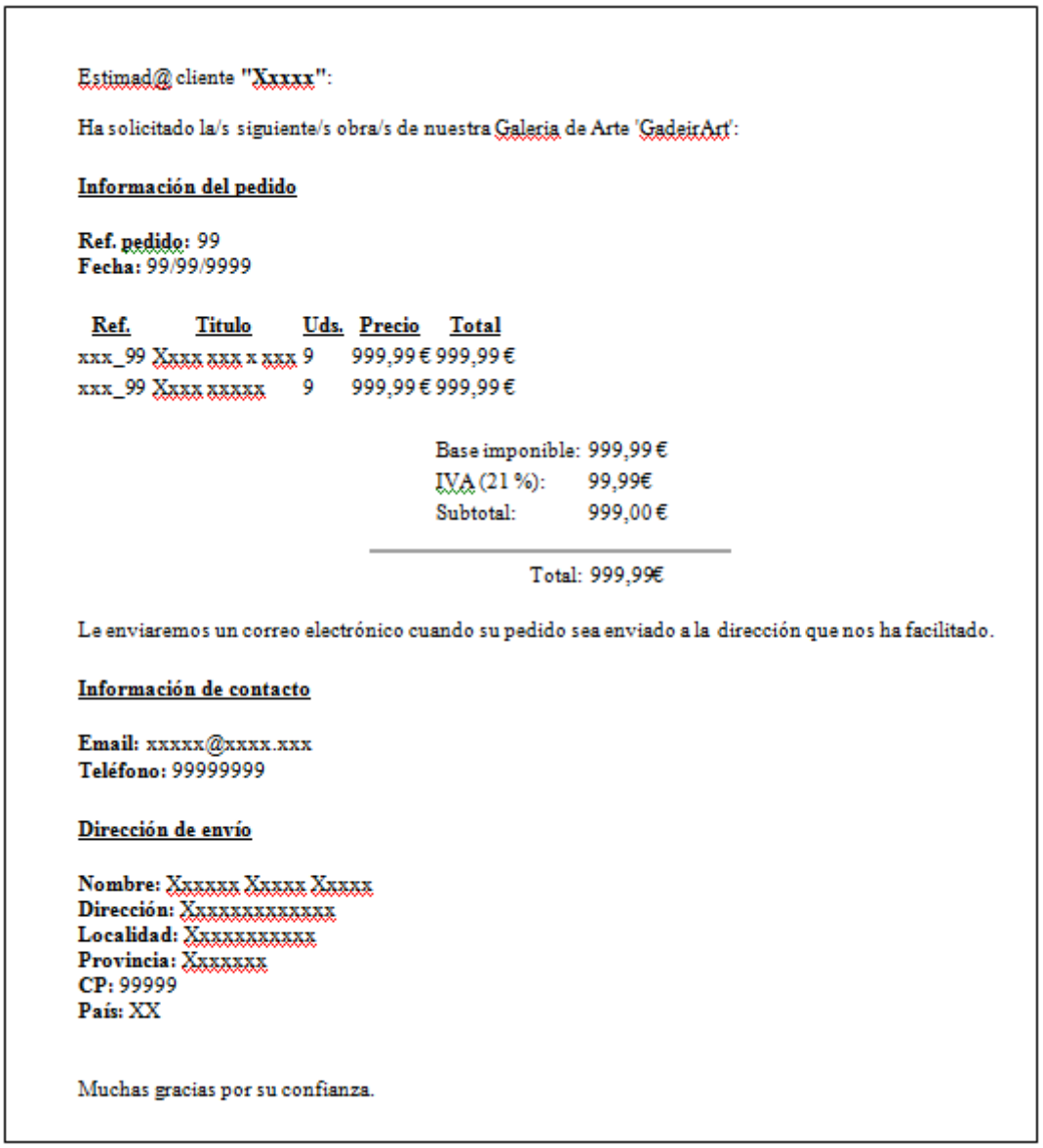


Figura 4.15: Mockup - Cesta de la compra

En la figura 4.21 podemos observar el diagrama de navegación de los diferentes menús para los usuarios[2] de la Galería online y, para que se vea de forma más clara, en la figura 4.22 vemos la parte correspondiente al *catálogo*.

La *cesta de la compra* actúa de diferente manera ya sea un usuario registrado[3] (figura 4.23) o un visitante (figura 4.24), veamos los diagramas de navegación para cada uno de ellos. Como se puede observar he marcado con un círculo rojo la parte que difiere.

[2] *Usuario*: Registrado o Visitante
[3] *Usuario registrado*: Cliente o Administrador



Estimad@ cliente "**Xxxxx**":

Ha solicitado la/s siguiente/s obra/s de nuestra Galeria de Arte 'GadeirArt':

Información del pedido

Ref. pedido: 99
Fecha: 99/99/9999

Ref.	Titulo	Uds.	Precio	Total
xxx_99	Xxxx xxx x xxx	9	999,99 €	999,99 €
xxx_99	Xxxx xxxxx	9	999,99 €	999,99 €

Base imponible: 999,99 €
IVA (21 %): 99,99€
Subtotal: 999,00 €

Total: 999,99€

Le enviaremos un correo electrónico cuando su pedido sea enviado a la dirección que nos ha facilitado.

Información de contacto

Email: xxxxx@xxxx.xxx
Teléfono: 99999999

Dirección de envío

Nombre: Xxxxxx Xxxxx Xxxxx
Dirección: Xxxxxxxxxxxxx
Localidad: Xxxxxxxxxxx
Provincia: Xxxxxxx
CP: 99999
País: XX

Muchas gracias por su confianza.

Figura 4.16: Mockup email - Confirmación del pedido al usuario registrado

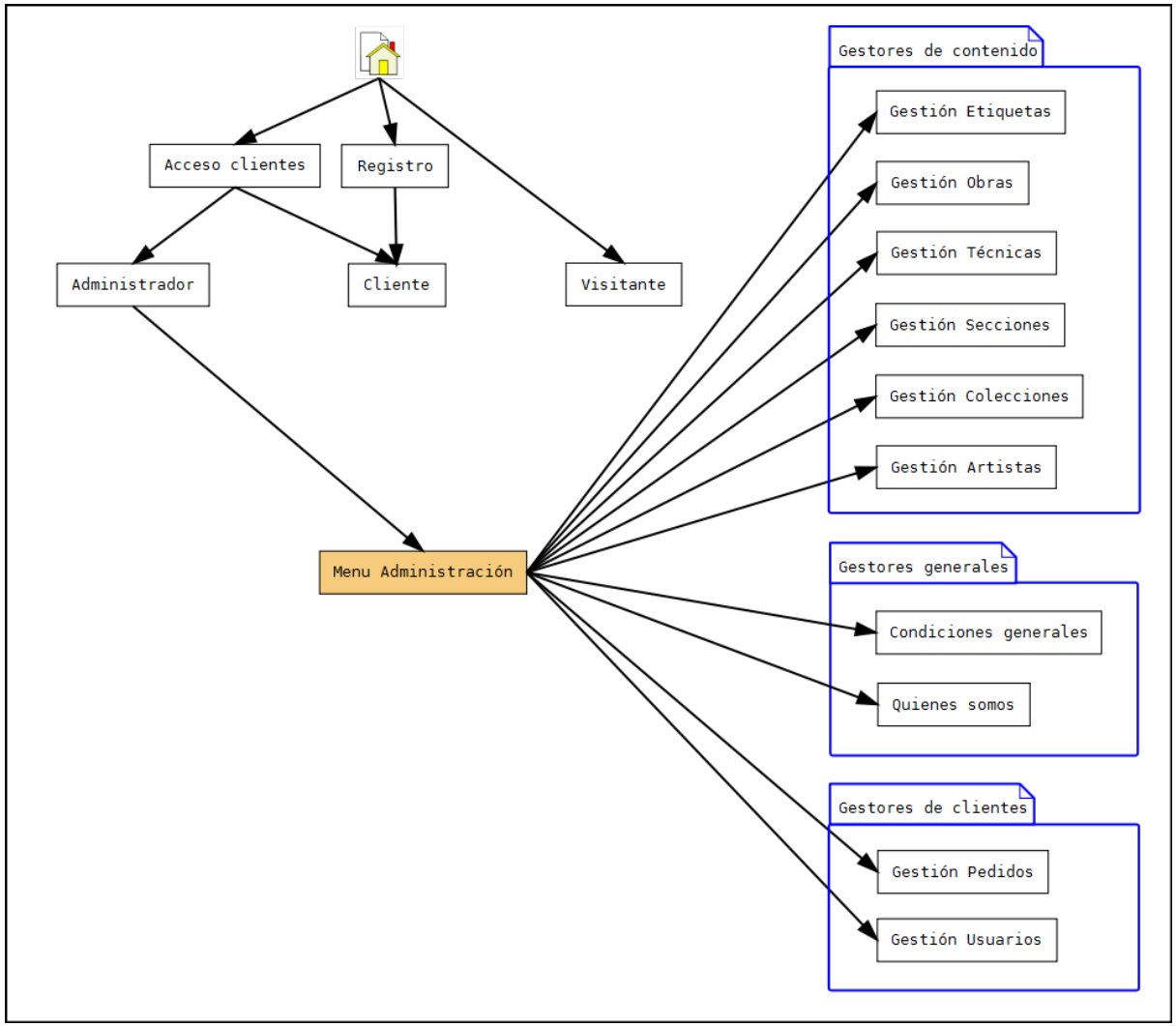


Figura 4.17: Diagrama de navegación para el administrador: Menú Administración

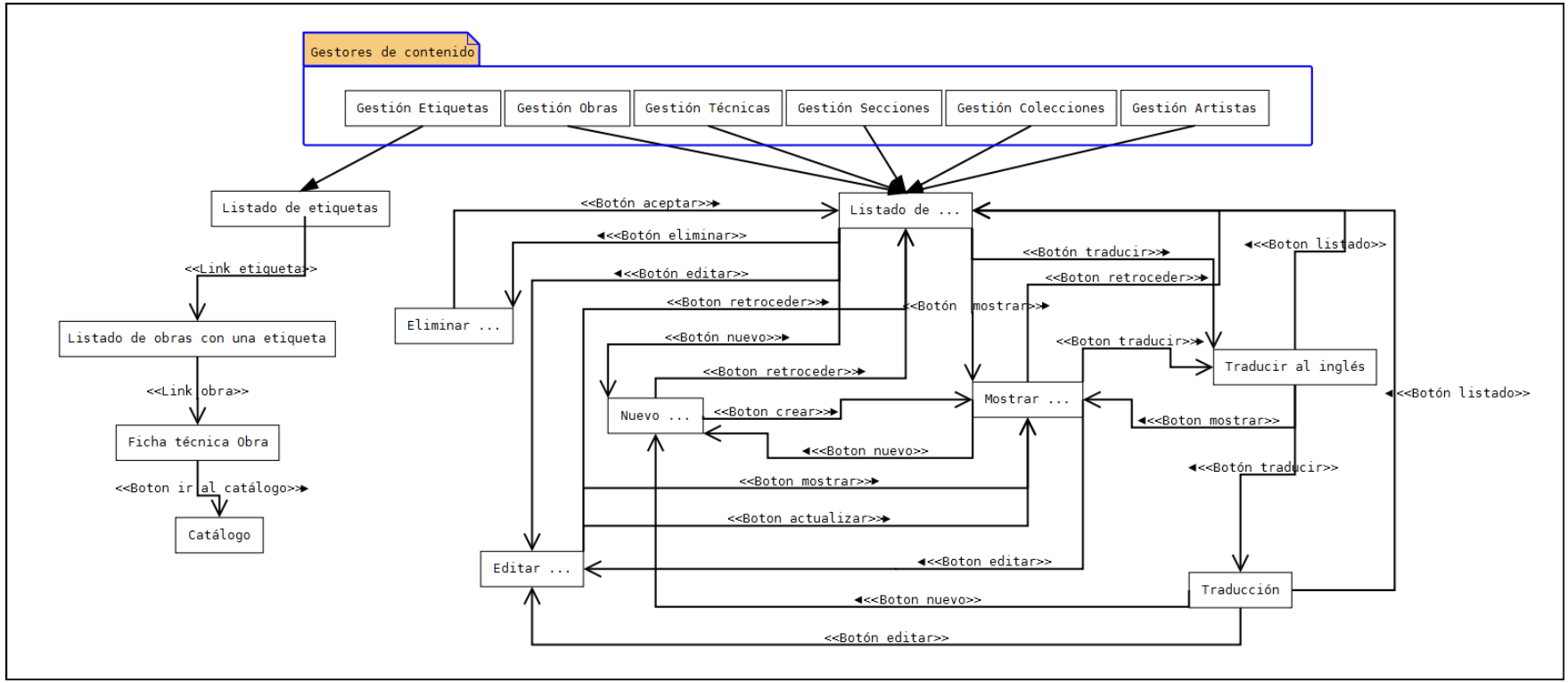


Figura 4.18: Diagrama de navegación para el administrador: Gestores de contenidos

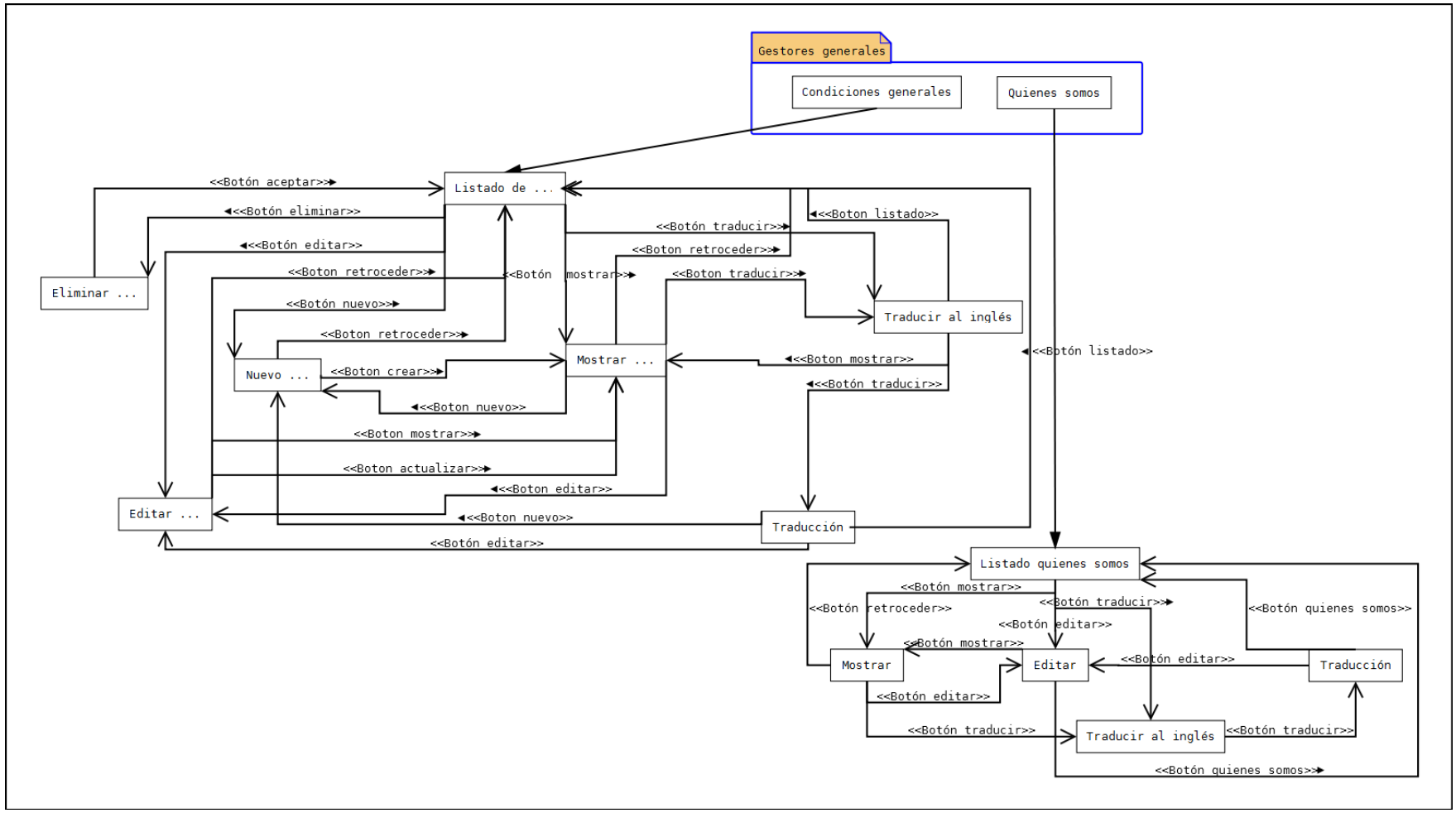


Figura 4.19: Diagrama de navegación para el administrador: Gestores generales

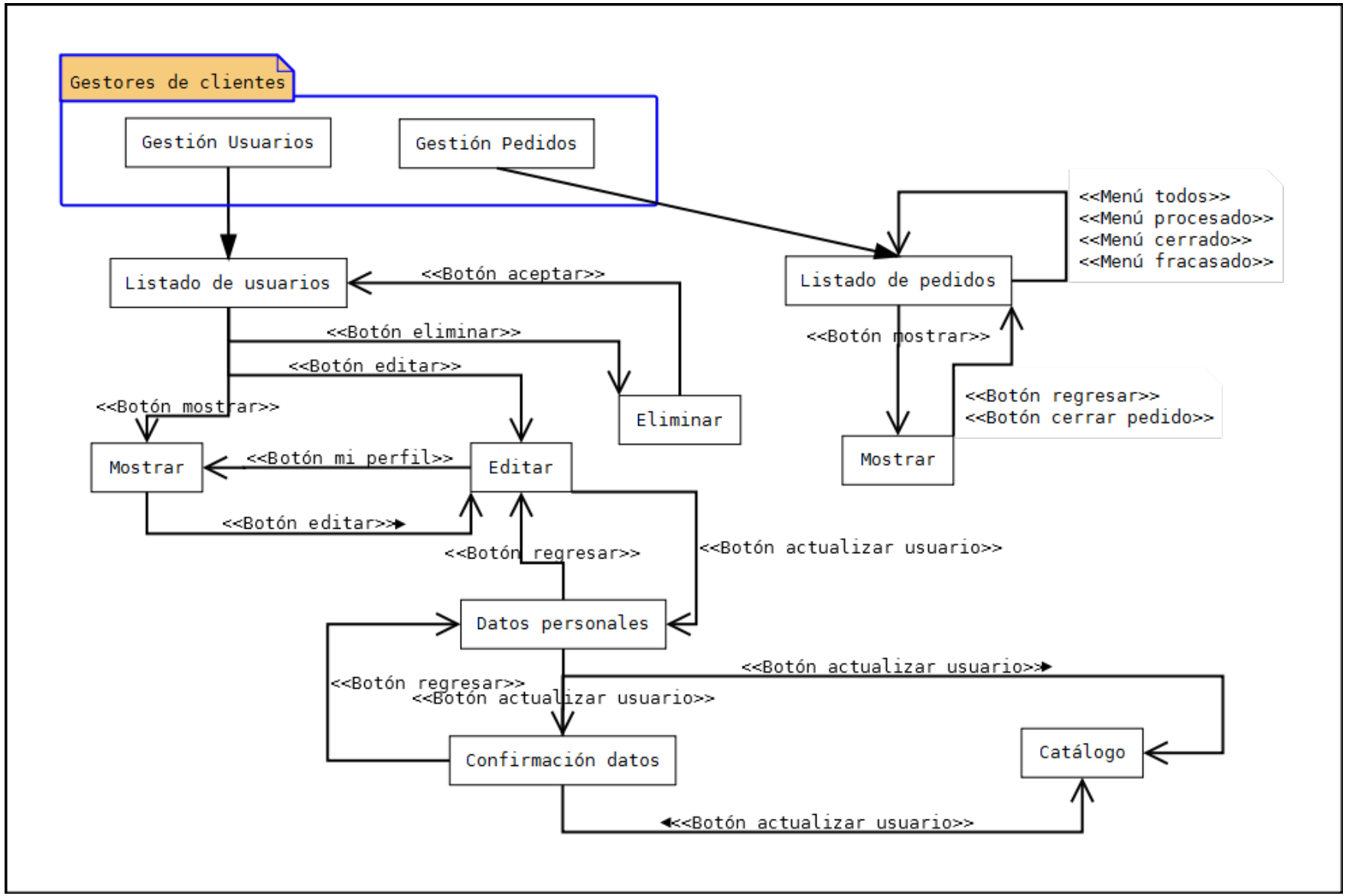


Figura 4.20: Diagrama de navegación para el administrador: Gestores de clientes

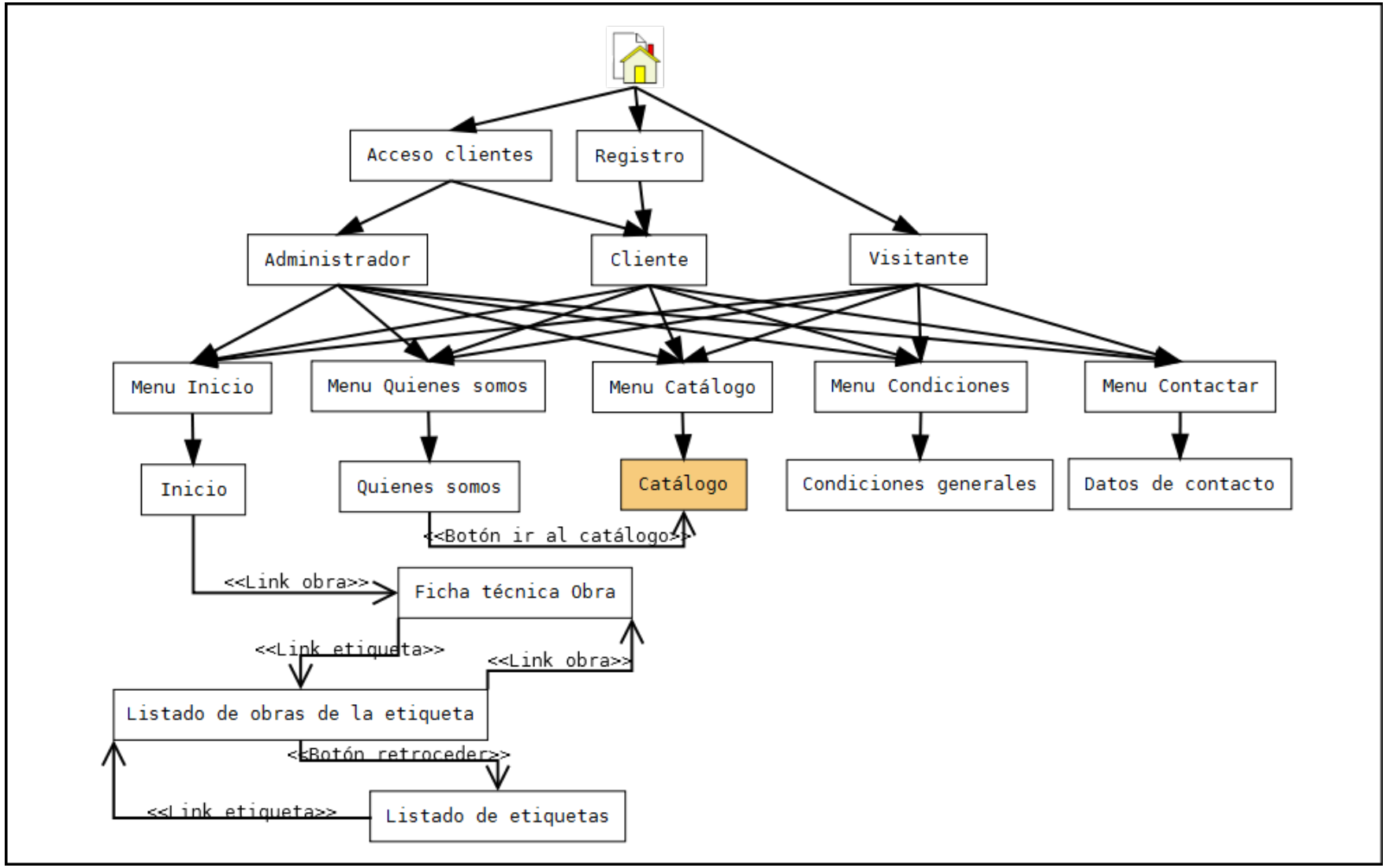


Figura 4.21: Diagrama de navegación para los usuarios: Menús

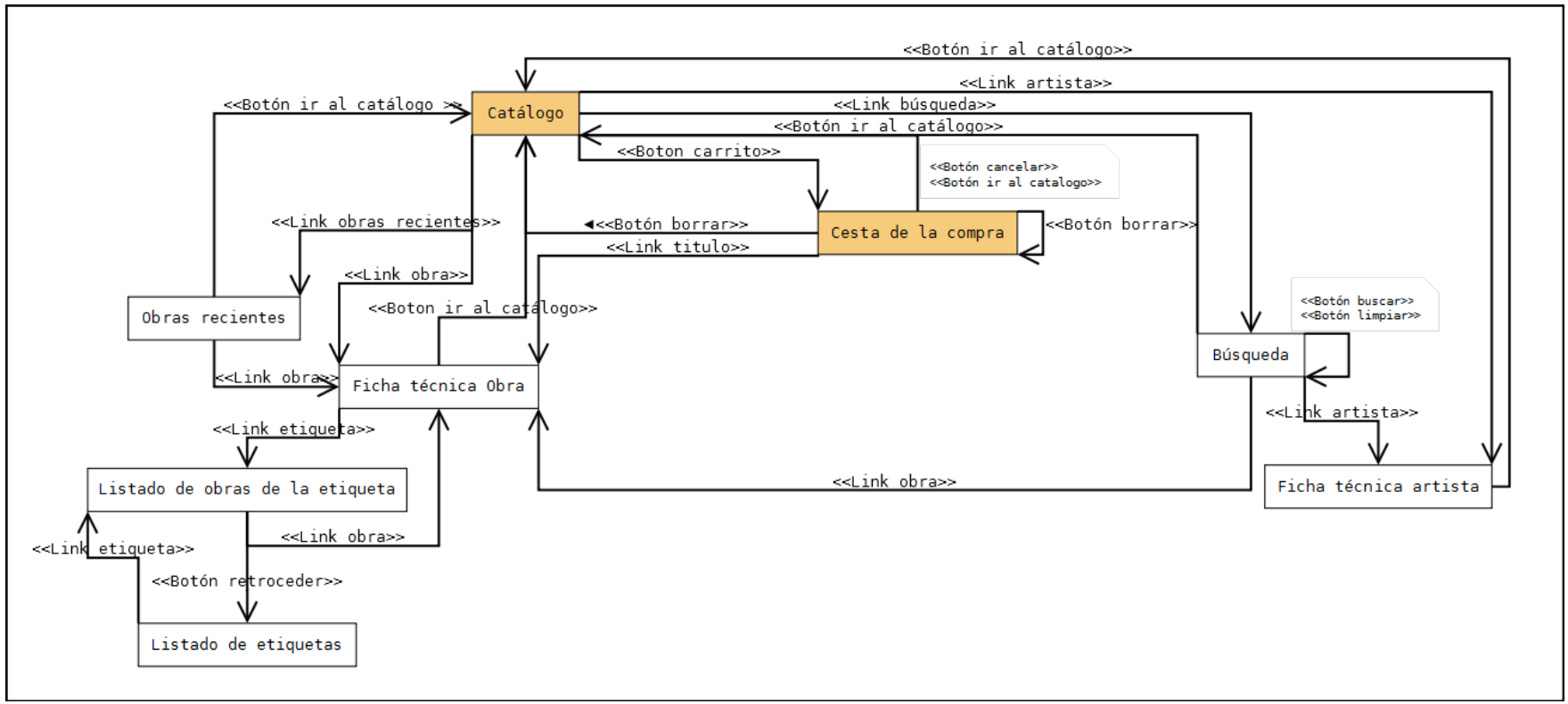


Figura 4.22: Diagrama de navegación para los usuarios: Catálogo

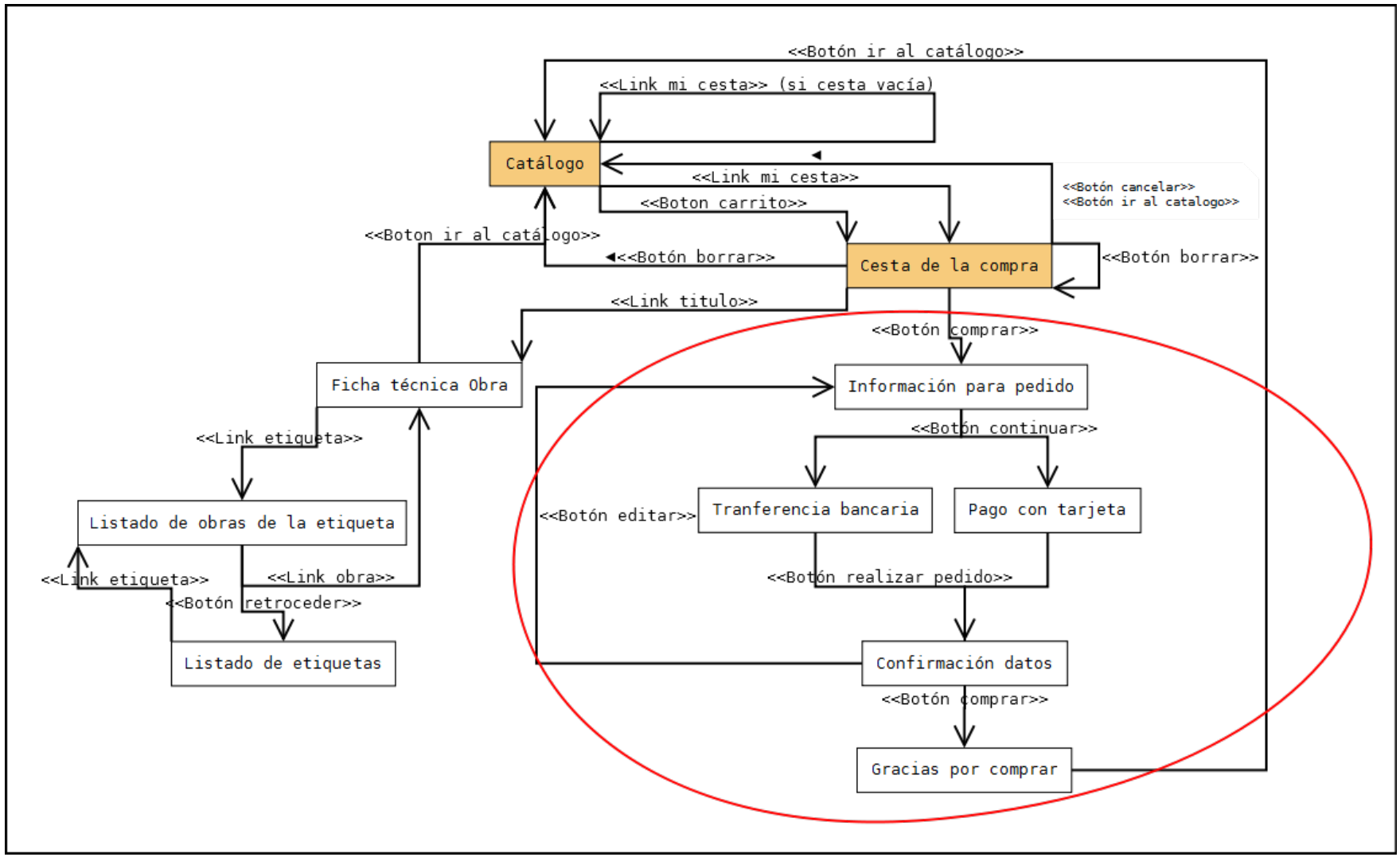


Figura 4.23: Diagrama de navegación para el cliente: Cesta de la compra

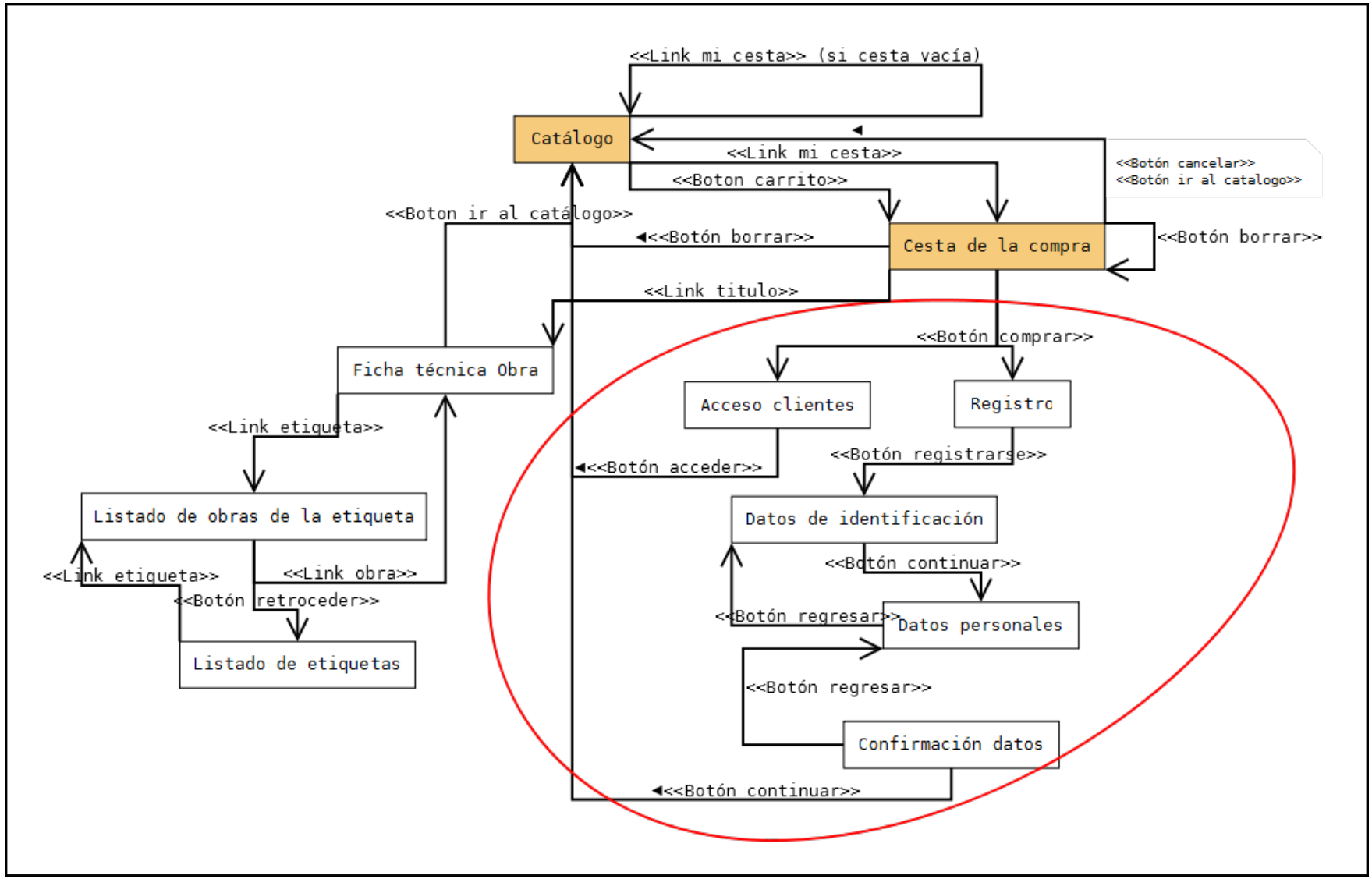


Figura 4.24: Diagrama de navegación para el visitante: Cesta de la compra

4.3. Diseño de datos

En esta sección se define la estructura física de datos que utilizará el sistema, a partir del modelo conceptual de clases mostrado en la sección 3.3.

Para ello, he teniendo presente los requisitos establecidos para el sistema y las particularidades del entorno tecnológico, de forma que se consiga un acceso eficiente de los datos.

4.3.1. Esquema de la BD

El esquema de una base de datos describe su estructura[4] en un lenguaje formal soportado por un *Sistema Administrador de Base de Datos (DBMS)* que en mi caso es *MySql*.

Podemos ver en la *Figura 4.25* representado el esquema de la base de datos del proyecto.

En el modelo de datos he tenido en cuenta requisitos básicos en el diseño de bases de datos como los de tener una estructura normalizada de las tablas, controlando la redundancia de la información, manteniendo la consistencia de datos, evitando las perdidas de información y manteniendo la integridad de los datos guardados.

Los nombres de las tablas se han elegido siguiendo las convenciones de *RoR*, donde *Active Record* obliga a que las tablas de la base de datos tengan el mismo nombre que las clases del modelo pluralizadas.

[4] *Estructura de la BD*: tablas, campos de cada tabla y relaciones existentes.

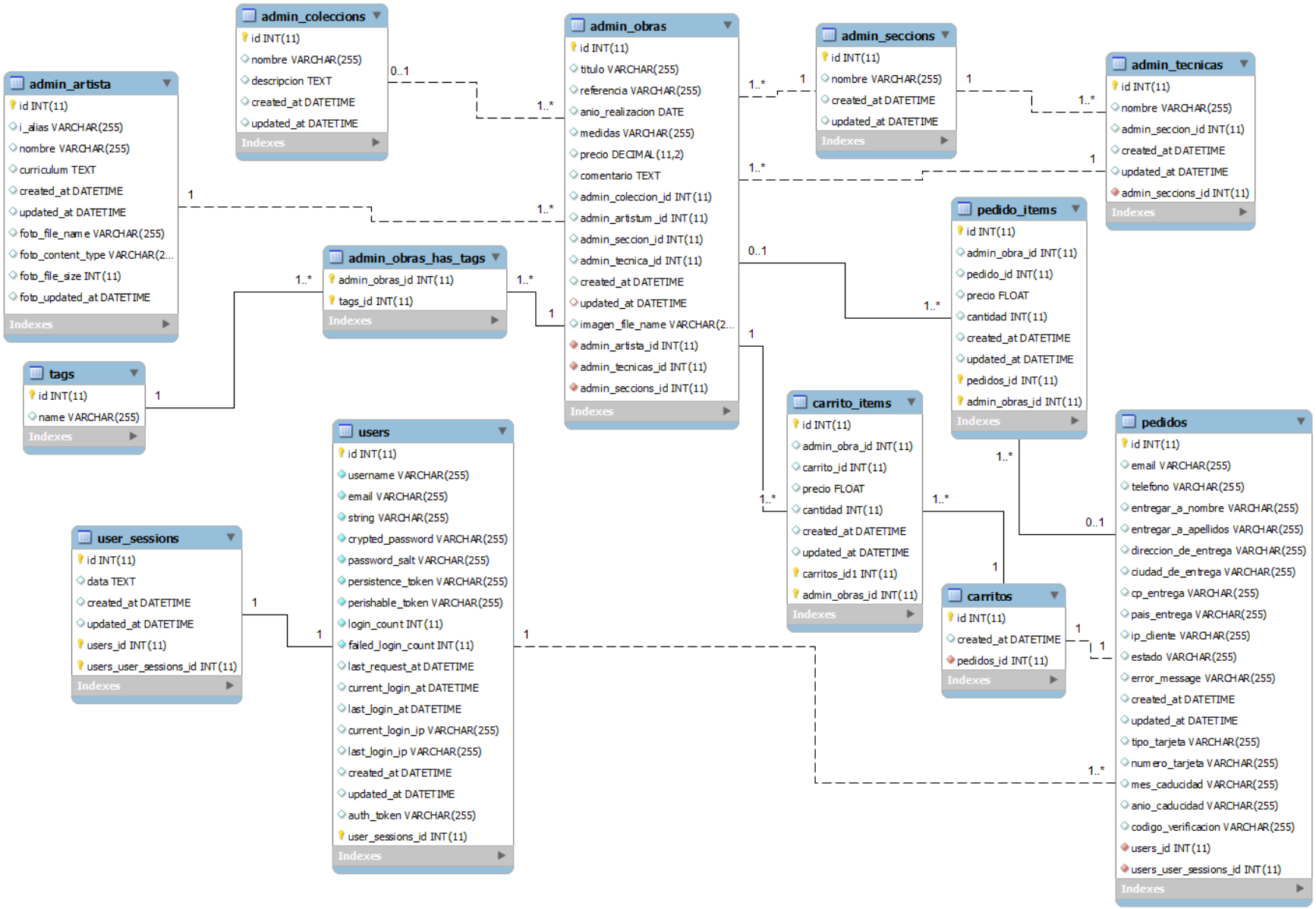


Figura 4.25: Esquema ERR de la Base de Datos

A continuación muestro, como ejemplo, el esquema de la base de datos escrito en lenguaje formal para una de las tablas. El esquema completo lo podemos encontrar en el fichero *\pitig\galeriaArte\db\schema.rb* del proyecto.

```
create_table "admin_obras", :force => true do |t|
    t.string   "titulo"
    t.string   "referencia"
    t.date     "anio_realizacion"
    t.string   "medidas"
    t.decimal  "precio",    :precision => 11, :scale => 2
    t.text     "comentario"
    t.integer  "admin_coleccion_id"
    t.integer  "admin_artistum_id"
    t.integer  "admin_seccion_id"
    t.integer  "admin_tecnica_id"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "imagen_file_name"
    t.string   "imagen_content_type"
    t.integer  "imagen_file_size"
    t.datetime "imagen_updated_at"
  end
```

Código 4.1: Esquema de BD para la tabla "admin_obras"

4.4. Diseño de componentes

4.4.1. Diagrama de secuencia (DSS)

Debido a que estoy usando una metodología ágil, en esta sección solo voy a mostrar, en la figura 4.26, un diagrama de secuencia ilustrativo del funcionamiento de las peticiones siguiendo el patrón MVC.

Los pasos son los siguientes:

1. El usuario introduce el evento, interactuando con la interfaz de usuario.
2. El *Controlador* recibe el evento y lo traduce en una petición al *Modelo* (aunque también puede llamar directamente a la *Vista*).
3. El *Modelo* (si es necesario) llama a la *Vista* para su actualización.

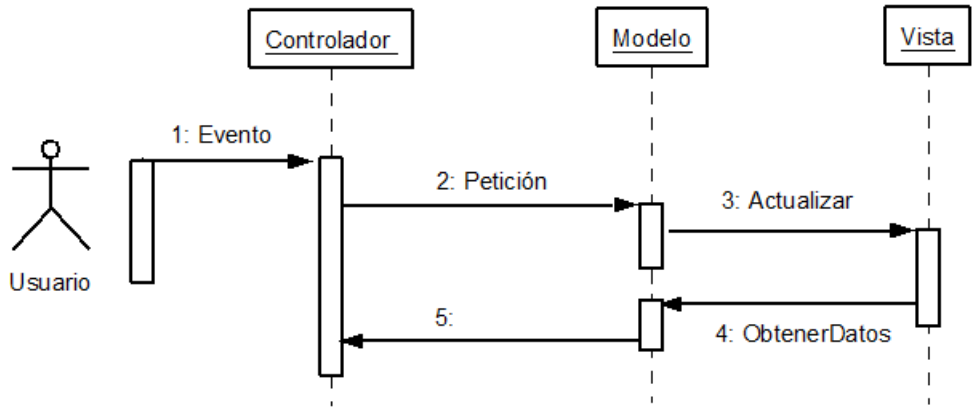


Figura 4.26: Diagrama de secuencia del patrón MVC

4. Para cumplir con la actualización la *Vista* puede solicitar datos al *Modelo*.

5. El *Controlador* recibe el control.

A modo de ejemplo veremos, en la *figura 4.27*, el diagrama de secuencia para el registro de un usuario en la galería. En él podemos observar como se construye un nuevo usuario dentro del sistema con los datos proporcionados por éste en la web.

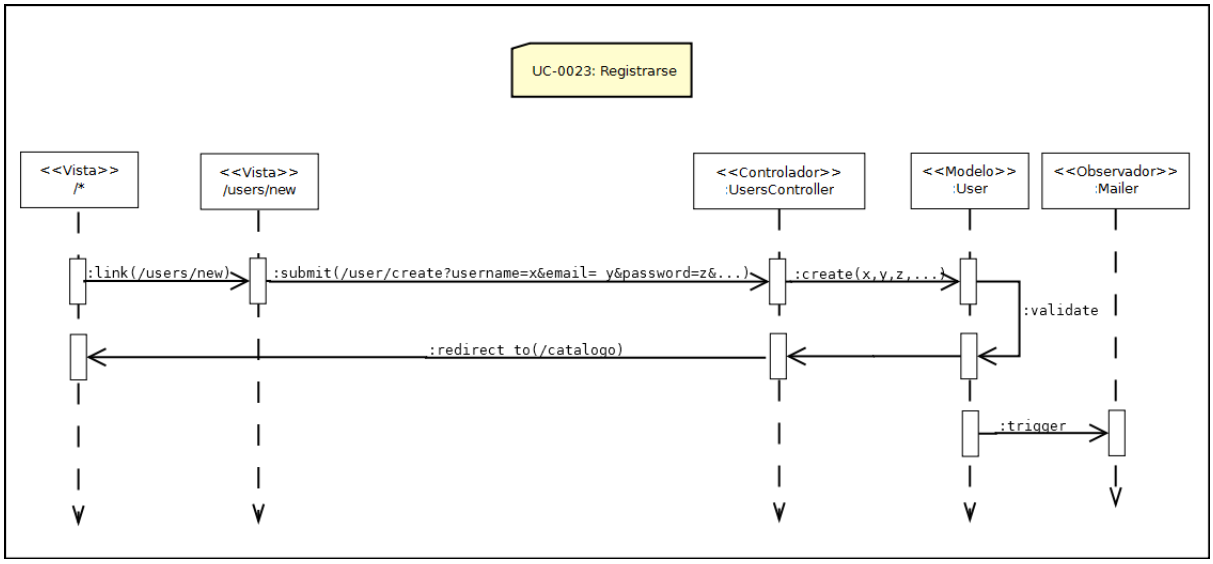


Figura 4.27: Diagrama de secuencia del CU-0023: Registrarse

4.4.2. Diagrama de clases de diseño

El *diagrama de clases* es el diagrama principal de diseño para un sistema. Se utiliza para representar la estructura estática de un sistema. Este contiene: las clases, atributos, operaciones y las relaciones de dependencia, generalización y asociación.

El *diagrama de clases de diseño* de la aplicación está representado en la figura 4.28.

4.5. Parametrización del software base

Al ver elegido *Ruby on Rails* como framework para el desarrollo, no he necesitado realizar ninguna parametrización para la correcta construcción de la aplicación.

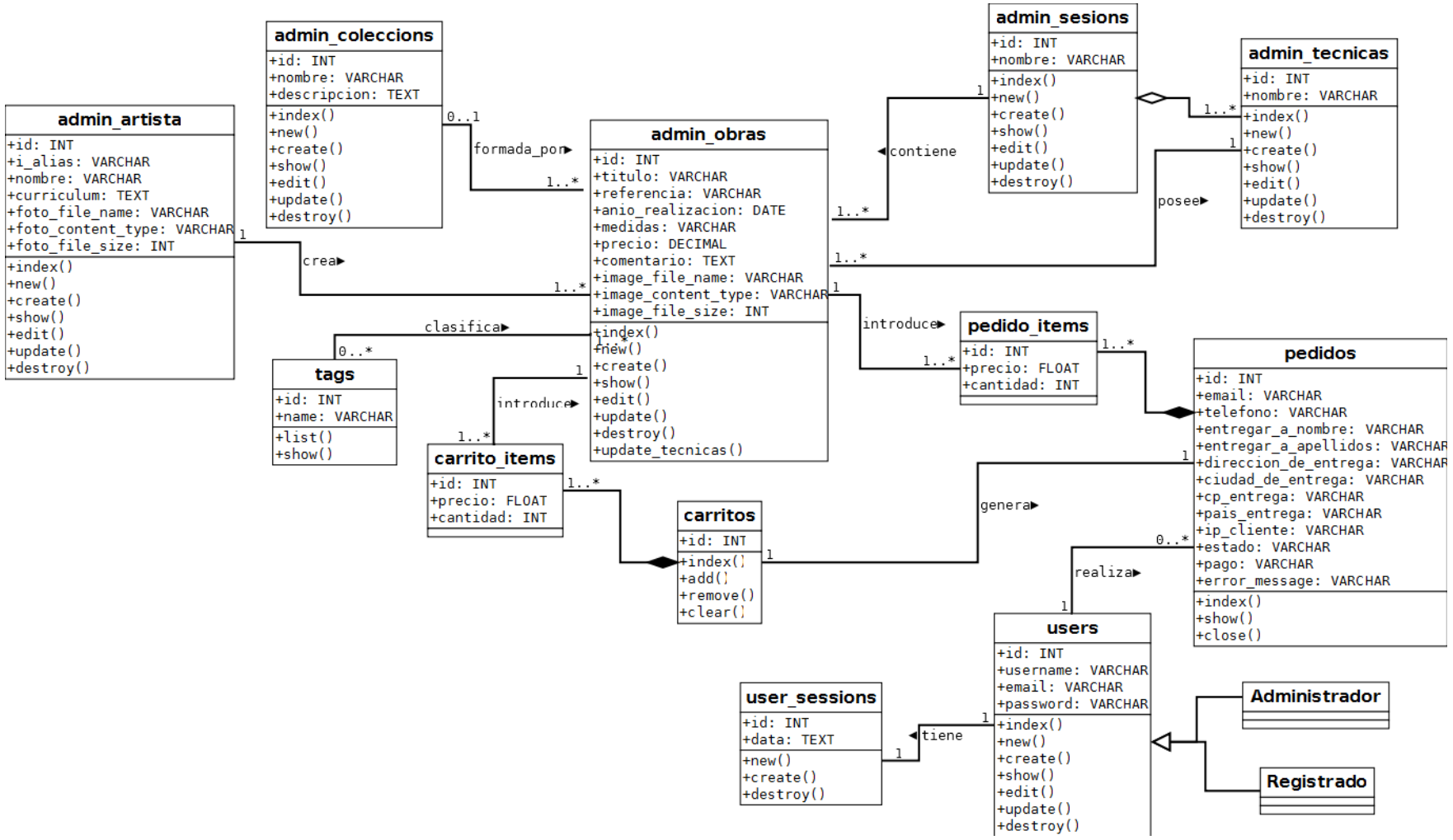


Figura 4.28: Diagrama clases de diseño

Capítulo 5

Implementación del Sistema

En el siguiente capítulo describiré los detalles más significativos de la fase de implementación de mi aplicación.

5.1. Entorno tecnológico

En esta sección se indica el marco tecnológico utilizado para la construcción del sistema.

5.1.1. Lenguajes de programación y framework

Como ya comenté en la *sección 3.6.1.2*, he seleccionado *Ruby* como lenguaje para implementar mi aplicación, ya que presenta una nueva forma de programar diferente a las estudiadas durante la carrera.

Para facilitarme el trabajo he elegido *Ruby on Rails* (ver *sección 3.6.2.1*), debido a que es un framework de código abierto para *Ruby* que sirve para desarrollar aplicaciones web y ayuda a programar siguiendo las *"buenas prácticas"* de programación.

El desarrollo de la aplicación lo realizaré aplicando el método tradicional, es decir, utilizaré un editor de texto, concretamente *"Gedit"* y el terminal para ejecutar los comandos de *Rails*.

En la aplicación también emplearé los siguientes lenguajes, los cuales se encuentran integrados de forma sencilla en *Ruby on Rails*.

Lenguaje HTML

Voy a emplear este lenguaje en las *Vistas* como veremos más adelante. En ellas este será el lenguaje principal. El código en *Ruby* integrado en las *vistas* va a ser traducido a *HTML* antes de ser interpretado por el navegador.

Los navegadores se encargan de mostrar a los usuarios las páginas web resultantes del código interpretado.

Lenguaje CSS

La idea principal del uso de CSS [W3C] es separar la *estructura* del documento de su *presentación*, con lo cual *HTML* define la estructura del documento y las *hojas de estilo* su aspecto visual.

A cada elemento *html* se le asignará un *id* o *class* y se le modificarán sus propiedades definiéndolas en un fichero *css* asociado al fichero *html* donde se encuentra dicho elemento.

5.1.2. Base de Datos

Aunque *SQLite*, SGBD por defecto, es fácil de administrar, mantener y personalizar, funciona bien solo en aplicaciones web que no cuenten con demasiada carga de acceso a la BD. Debido a esto, he preferido utilizar para el desarrollo de mi aplicación el SGBDR ***MySQL*** (Open Source).

El servidor *MySQL* fue desarrollado para manejar grandes bases de datos mucho más rápido que las soluciones existentes y aunque se encuentra en desarrollo constante, ofrece un conjunto rico y útil de funciones. Su ***conectividad, velocidad, seguridad***, y su ***facilidad de uso*** hacen de *MySQL* un servidor bastante apropiado para acceder a bases de datos en Internet.

5.1.3. Herramientas de desarrollo

A continuación voy a describir cada una de las herramientas que he utilizado en el desarrollo de mi proyecto. Todas ellas han sido bastante útiles y fáciles de utilizar.

- **Adobe Reader**: Es un ***lector estándar*** que me ayuda a visualizar e imprimir los documentos PDF, puede obtenerse en `www.adobe.es`.

- **Assembla**: Es una herramienta web que me permite el ***control de desarrollo*** y la ***coordinación de las actividades*** de mi proyecto. Ofrece elementos para la administración como: seguimiento de defectos y tareas, repositorios, control de versiones, control del tiempo empleado y Scrum.

  Para alojar mi proyecto he utilizado uno de los sistema de control de versiones que posee Assembla: "*Subversion*"(5.1.3), ya que de entre los que ofrece es el que mejor sabía manejar.

  Su página web es: `http://www.assembla.com/`.

- **Balsamiq Mockups**: Aplicación AIR pensada para ayudarnos a crear de manera fácil y rápida nuestros wireframes[1]. Lo consigue a través de un sistema de drag-and-drop de componentes predefinidos.

  Con este software he desarrollado los diferentes ***Mockups*** de la aplicación.

  Su página web es: `http://www.balsamiq.com/`.

- **Dia v.0.97**: Es un ***editor de diagramas*** con las herramientas necesarias para crear o modificar gráficos. Está concebido de forma modular, con diferentes paquetes de plantillas (UML, Cisco computer, diagramas de lógica, ...) para diferentes necesidades. Es software libre GNU con Licencia General Pública (GPL).

[1] *Wireframe*: Boceto que define el contenido y la funcionalidad de una web.

Podemos obtener buenos resultados con él en poco tiempo, ya que su utilización es muy fácil, flexible e intuitiva. Sus datos pueden ser exportados a diferentes formatos, como el formato gráfico ".png", ideal para incluir en proyectos.

- **Dropbox v.1.6.11**: Es un ***servicio de alojamiento de archivos*** multiplataforma en la nube, operado por la compañía *Dropbox*. Permite a los usuarios almacenar y sincronizar archivos en línea y entre computadoras y compartir archivos y carpetas con otros. Para el proyecto he utilizado la versión gratuita, donde he almacenado las imágenes de los artistas y obras de la Galería de Arte.

  La versión gratuita se puede descargar desde: `https://www.dropbox.com/`.

- **Firebug**: Es un complemento del gran navegador web *Firefox* que me permite ver, ***depurar*** y probar todo lo que tenga que ver con el lado del cliente: HTML, Javascript, CSS, pudiendo visualizar su efecto en la página en tiempo real.

- **gedit**: Es un ***editor de textos*** compatible con UTF-8 para GNU/Linux. Enfatiza la simplicidad y facilidad de uso e incluye herramientas para la edición de código fuente y textos estructurados, como lenguajes de marcado.

  gedit me proporciona un adecuado coloreado de sintaxis tanto en los archivos .rb (puro ruby) como en los .html.erb (Rhtml) y abre múltiples archivos en pestañas, suficiente para una cómoda edición de código.

  Distribuido bajo las condiciones de la licencia GPL, gedit es software libre y es el editor predeterminado de GNOME.

- **Gestor de versiones "Subversión"**: Un gestor de versiones es un ***servidor de almacenamiento de ficheros*** que almacena las diferencias en un fichero por cada envío. De esta forma puedo mantener un historial de todos los cambios que se realicen en el fichero a lo largo de su desarrollo y la última versión será la suma de todos los cambios.

En el desarrollo de software resulta útil tener un historial del trabajo que se está realizando y poder volver a una copia anterior con relativa facilidad. También me servirá como copia de seguridad del trabajo realizado.

El gestor de versiones lo puede encontrar en las siguientes direcciones:

- Galería de arte: `https://www.assembla.com/code/pitig/subversion/nodes`
- Artista: `https://www.assembla.com/code/pitig_artista/subversion/nodes`

- **Git**: Es un software de ***control de versiones*** que utilizaré para el despliegue de la aplicación en *Heroku*.
- **Heroku** [Heroku]: Es un servicio gratuito[2] de ***hosting en la nube*** para el lenguaje de programación *Ruby*, aunque también da soporte a otros lenguajes como Java, PHP... Su S.O. base es Debian.

  Este servicio lo he utilizado para el ***despliegue online*** de mis aplicaciones, pudiendo observar el de la *Galería de Arte* en `http://gadeirart.herokuapp.com/` y el de la *Artista* en `http://artista.herokuapp.com`.

  La página web principal de Heroku es: `http://www.heroku.com/`.

- **MySql WorkBrench**: Es una ***herramienta visual de diseño de bases de datos*** que integra desarrollo de software, administración y diseño de bases de datos, creación y mantenimiento para el sistema de base de datos MySQL. Con ella he realizado el diagrama ERR de la base de datos de mi aplicación. Es open source y podemos obtenerlo en: `http://www.mysql.com/products/workbench/`.

- **Navegadores**: Emplearé diferentes navegadores para poder probar y depurar mi aplicación. Entre ellos: Firefox, Chromium, Internet Explorer.

[2] *Gratuito* para aplicaciones de poco consumo, lo cual es bastante bueno para poder desarrollar y lanzar la aplicación sin pagar.

- **Nero Burning ROM**: Programa para ***grabar todo tipo de archivos*** en CD o DVD. Abarca todo lo que es datos, videos, sonido . . . Podemos obtener una versión de prueba en: http://www.nero.com.

- **OpenProj** [Leonardo Lemus, Jorge; Navas Muños, Jenniffer]: Es una aplicación gratuita y de código abierto que puede ser utilizada bajo licencia *CPAL («Common Public Attribution License»)*.

  OpenProj™

  Esta herramienta está pensada para la ***gestión de proyectos y tareas*** y nos permite generar gráficos como:

  - Diagramas Gantt.
  - Diagramas PERT.
  - Diagramas WBS.
  - Diagramas RBS.

  En concreto, con este software he desarrollado el ***Diagrama de Gantt*** del proyecto.

- **Pingdom**: Herramienta online que permite medir ***la velocidad de carga*** de las páginas web, además genera un gráfico bastante sencillo de entender donde se ve la evolución de carga en segundos de cada uno de los archivos de la web, el cual permitirá saber cuál es el origen de una posible carga lenta y a qué archivo se debe. Su url es http://tools.pingdom.com/fpt/.

- **StatSVN**: Herramienta para ***analizar los repositorios de subversion***. Con ella se pueden obtener una serie de reportes interesantes que muestran la evolución de cada repositorio. Bastante sencilla de utilizar. Entre los reportes tenemos:

  - Total de líneas de código en el tiempo.
  - Líneas de código por desarrollador.
  - Actividad por día y hora.
  - Total de archivos y tamaño promedio.
  - Archivos con mas revisiones.

  StatSVN

  Su página web es: www.statsvn.org.

- **WinEdt** [Simonic, Aleksander]: Para redactar la presente memoria del proyecto he empleado este potente y versátil ***editor de texto*** para Windows, creado por el matemático *Aleksander Simonic*, con una fuerte predisposición hacia la creación de documentos LaTeX[Burgos Pintos, Aris; Cornejo Barrios, Alicia; Gámez Mellado, Antonio; Otros].

Ha sido específicamente diseñado y configurado para integrarse perfectamente con un sistema de *TeX* como **MiKTeX**, creado y mantenido por *Christian Schenk*.

Se puede descargar en la dirección: http://www.winedt.com/.

5.1.4. Componentes

Lo más común en una aplicación escrita en *Rails* es que se reutilize código creado por otros desarrolladores. Este código puede ser distribuido en ***gemas***, las cuales podemos decir, que son librerías que nos aportan nuevas funcionalidades para nuestro desarrollo.

Éstas se instalan o desinstalan mediante el *gestor de paquetes* ***RubyGems***, el cual proporciona:

- un formato estándar y autocontenido llamado ***gem*** para poder distribuir programas o librerías en *Ruby*,
- una herramienta destinada a gestionar la instalación de éstos y
- un servidor para su distribución.

En otras palabras, ***RubyGems*** nos provee de una aplicación llamada ***gem*** que permite *instalar, desinstalar* y *consultar* sobre las librerías o gemas que tengamos instalada para implementar en el desarrollo.

La *configuración de gemas* se realizan en el fichero *"Gemfile"* que se encuentra en el *directorio raíz* de la aplicación. A continuación, a modo de ejemplo, muestro parte del contenido del fichero *"Gemfile"* de la aplicación.

```
source 'http://rubygems.org'

gem 'rails', '3.0.9'
```

```
# Bundle edge Rails instead:
# gem 'rails', :git => 'git://github.com/rails/rails.git'

# Usamos MySql
gem 'mysql', :require => 'mysql'

# Gema para la subida y tratamiento de imagenes
gem 'paperclip', '2.3.16'

# Gema para tratar paperclip en heroku mediante Dropbox
gem 'paperclipdropbox', '1.0.5'

# Gema para paginar las vistas
gem 'will_paginate', '~> 3.0.pre2'
....
```

Código 5.1: Extracto del fichero "Gemfile"

En el fichero podemos ver que la mayoría de las gemas se instalan desde el repositorio de gemas `http://rubygems.org`, aunque también se pueden instalar desde un repositorio Git o desde una ruta. Junto al nombre de la gema le he indicado la versión que deseo instalar.

Rails posee la herramienta ***Bundler*** para gestionar de forma sencilla y eficaz las dependencias de gemas configuradas en el fichero *"Gemfile"*. Ésta se asegura de que la aplicación siempre cuente con las gemas necesarias para funcionar sin problemas. Mediante la orden:

```
$>bundle install
```

se instalan todas las dependencias necesarias, guardándolas en el fichero *"Gemfile.lock"* del *directorio raíz*. Este fichero contiene una lista exhaustiva de todas las gemas que necesita la aplicación junto con su número de versión, es decir, las que he declarado en *"Gemfile"* y todas sus dependencias.

Veamos a modo de ejemplo un extracto de mi fichero *"Gemfile.lock"*.

```
GEM
  remote: http://rubygems.org/
  specs:
    ....
    activerecord (3.0.9)
      activemodel (= 3.0.9)
      activesupport (= 3.0.9)
```

```
      arel (~> 2.0.10)
      tzinfo (~> 0.3.23)

    ....
    rails (3.0.9)
      actionmailer (= 3.0.9)
      actionpack (= 3.0.9)
      activerecord (= 3.0.9)
      activeresource (= 3.0.9)
      activesupport (= 3.0.9)
      bundler (~> 1.0)
      railties (= 3.0.9)
        ....

PLATFORMS
  ruby

DEPENDENCIES
  ....
  nokogiri (= 1.4.4)
  paperclip (= 2.3.16)
  paperclipdropbox (= 1.0.5)
  rails (= 3.0.9)
  sunspot_rails (~> 1.2.1)
  tzinfo
  will_paginate (~> 3.0.pre2)
```

Código 5.2: Extracto del fichero "Gemfile.lock"

En el *anexo C "Gemas"* podrá encontrar descrita cada una de las gemas que he utilizado para el desarrollo de la aplicación.

5.2. Código fuente

Para poder comenzar mi proyecto [Hellsten, Christian; Laine, Jarkko], lo primero que me hizo falta fue instalar *Ruby on Rails* y todos los paquetes que están relacionados con este último que, por defecto, suelen instalarse con él.

Una vez instalados y tras ejecutar el comando

```
$>rails new galeriaArte
```

en la consola, se construyó la estructura de directorios y los ficheros de configuración de mi aplicación.

A continuación explicaré brevemente la funcionalidad de cada uno de ellos:

- **app** En este directorio se organizan los **componentes de la aplicación** y es donde tiene lugar la mayor parte del proceso de implementación. Lo forman los siguientes subdirectorios:
  - **controllers:** Aquí podemos encontrar todos los controladores, es decir, el comportamiento lógico de la aplicación.
  - **helpers:** Son métodos comunes de la aplicación que contienen las clases que ayudan a los controladores. Se crean por defecto cuando se crea un *controller*.
  - **mailers:** En esta carpeta se define la clase *UserMailer* cuya función es específicamente el envío de correos.
  - **models:** Contiene las clases que modelan y empaquetan los datos almacenados en la base de datos de la aplicación. En ellas se definen las relaciones entre entidades, las validaciones de datos . . . controlándose la integridad de los datos.
  - **views:** En esta carpeta se guardan las descripciones de las vistas de la aplicación, es decir, las páginas *RHTML* de las que está compuesta. Su estructura está subdivida por entidades poseyendo cada una sus propias páginas para poder operar con el contenido de BD.

    Existe una carpeta llamada *layout* que contiene la plantilla principal de la aplicación. Esta modela las partes comunes de las vistas como: el encabezado, el menú principal, el pie de página. . . .
- **config** Contiene el código relativo a la **configuración de la aplicación**, siendo los aspectos más destacados la configuración de:
  - la base de datos (*database.yml*),
  - las rutas y los recursos de la aplicación (*routes.rb*) y
  - los tres entornos que presenta *Rails*: desarrollo, pruebas y producción (contenidos dentro del subdirectorio `\environments`).
    - **Entorno de desarrollo (*development*):** Optimiza la productividad del desarrollador. Las caches apenas operan, así que los cambios en el código de la aplicación se aprecian rápidamente, sin tener que recompilar o volver a desplegar nada. Sólo hay que recargar la página del navegador.
    - **Entorno de pruebas (*test*):** Optimizado para ejecutar pruebas unitarias, funcionales y de integración. Cada vez que se ejecuta una prueba, la base de datos se limpia de todos sus datos y *Ruby on Rails* se encarga de poblarla con datos de prueba antes de cada test, a través de sus fixtures[3].
    - **Entorno de producción (*production*):** Es donde se despliega la aplicación final. Este entorno está optimizado para rendimiento.

[3]Los *fixtures* son los datos de carga.

- **db** Este directorio alberga un subdirectorio llamado *migrate* que contiene las **migraciones de *ActiveRecord***, las cuales se encargarán de la creación o eliminación de tablas y de la inclusión o eliminación de campos en ellas, y los ficheros del **esquema de base de datos**.
- **doc** *RoR* posee **Rubydoc**, una interfaz a partir de la cual es posible generar automáticamente documentación sobre la aplicación a partir de los comentarios recogidos en el código y que se almacenarán en este directorio.
- **lib** Aquí se almacenan las librerías adicionales que son utilizadas en el proyecto.
- **log** Contiene las trazas de ejecución, que resultan especialmente útiles para tareas de depuración.
- **public** Almacena los recursos internos de la aplicación como imágenes, hojas de estilos, javascripts ...
- **script** Contiene el script *rails* que sirve para iniciar la aplicación.
- **solr** Contiene los ficheros de configuración necesarios para la realización de las búsquedas dentro de la aplicación.
- **test** Aquí se almacenan las correspondientes pruebas unitarias, funcionales y de integración llevadas a cabo junto con sus *fixtures*.
- **tmp** Dedicado al almacenamiento de aquellos archivos temporales que puedan estar relacionados con el funcionamiento de la aplicación.
- **vendor** Para aquellas librerías externas que extiendan el funcionamiento básico de *Rails*.

Además de estos directorios, destacan dos ficheros:

- **Rakefile:** Su finalidad es similar al *Makefile de GNU*. En él se configuran los aspectos orientados a la construcción de la aplicación, su distribución en paquetes o los tests a realizar. Esta configuración es utilizada por el comando ***rake*** (hecho a similitud del *make de GNU*).
- **Gemfile:** En este fichero se indican las gemas necesarias para el funcionamiento de mi aplicación. Este es procesado por el *Bundler* que se encarga de gestionar las dependencias de las gemas en *Rails*.

Esqueleto para la aplicación

Una de las utilidades que ofrece *Rails* y que he aprovechado para la implementación es el **Scaffolding**, el cual sirve para crear un *esqueleto* funcional, que genera una interfaz

web que permite llevar a cabo las operaciones básicas CRUD sobre una tabla especificada de la BD.

Por cada elemento que representa una tabla en la base de datos, al ejecutar el script **Scaffold** (figura 5.1) se genera:

- una clase para la creación/migración de la tabla en la BD en *db\migrate*,
- una clase para el modelo en *app\models*,
- una clase para los test unitarios en *test\unit*,
- un fichero para la carga de datos en *test\fixtures*,
- una clase para el controlador que implementa las funciones CRUD en *app\controllers*,
- una vista para cada una de las funciones del controlador en *app\views\<nombre_tabla>*,
- una clase para los test funcionales en *test\functional*,
- un módulo para los helpers en *app\helpers* y
- una hoja de estilo, la cual he eliminado ya que creo las mías propias para el proyecto, en *public\stylesheets*.

A partir del esqueleto y para cada uno de los sprint, he ido implementando lo necesario para obtener mi aplicación final.

5.3. Implementación del sistema

En esta sección describo las implementaciones más destacadas realizadas en el sistema.

5.3.1. Implementación del Modelo

El **Modelo** es el componente del *patrón de diseño MVC* encargado del acceso a los datos.

Llamaremos "modelo" a una clase que trabaja en relación a una tabla de la base de datos, las instancias de esta clase son "instancias del modelo" y el modelo encapsula la "*lógica de negocios*".

Siguiendo el *patrón de diseño* ***ActiveRecord*** he programado la *lógica de negocio* de mi aplicación.

Este patrón me ayuda a realizar todas las tareas de la base de datos: *Definición de Datos* y *Manipulación de Datos* a través de sus capacidades de metaprogramación, sin tener que escribir sentencias SQL.

```
    invoke  active_record
    create    db/migrate/20120321195714_create_admin_obras.rb
    create    app/models/admin/obra.rb
 identical    app/models/admin.rb
    invoke    test_unit
    create      test/unit/admin/obra_test.rb
    create      test/fixtures/admin/obras.yml
     route  namespace :admin do resources :obras end
    invoke  scaffold_controller
    create    app/controllers/admin/obras_controller.rb
    invoke    erb
    create      app/views/admin/obras
    create      app/views/admin/obras/index.html.erb
    create      app/views/admin/obras/edit.html.erb
    create      app/views/admin/obras/show.html.erb
    create      app/views/admin/obras/new.html.erb
    create      app/views/admin/obras/_form.html.erb
    invoke    test_unit
    create      test/functional/admin/obras_controller_test.rb
    invoke    helper
    create      app/helpers/admin/obras_helper.rb
    invoke      test_unit
    create        test/unit/helpers/admin/obras_helper_test.rb
    invoke  stylesheets
 identical    public/stylesheets/scaffold.css
```

Figura 5.1: Ejecución del *script Scaffold* para la tabla "admin_obras"

5.3.1.1. Definición de datos

La definición de la base de datos la he realizado utilizando migraciones, las cuales son clases *Ruby* que *ActiveRecord* se encarga de interpretar y traducir en las secuencias SQL necesarias según el motor de base de datos con el que se esté trabajando.

Las migraciones contienen los métodos necesarios para la creación de la tabla con todos sus campos, así como su eliminación, y se pueden localizar en el directorio *"db\migrate"* de mi aplicación con el siguiente nombre:

<fecha y hora de creación>_create_<nombre de la tabla>.rb

A continuación muestro, como ejemplo, el fichero de migración de la tabla "admin_obras".

```
class CreateObras < ActiveRecord::Migration
  def self.up
    create_table :admin_obras do |t|
      t.string     :titulo
      t.string     :referencia
      t.date       :anio_realizacion
      t.string     :medidas
      t.decimal    :precio, :precision => 11, :scale => 2
      t.text       :comentario
      t.references :admin_coleccion
      t.references :admin_artistum
      t.references :admin_seccion
      t.references :admin_tecnica

      t.timestamps
    end

    Admin::Obra.create_translation_table! :titulo => :string,
          :comentario => :text
  end

  def self.down
    drop_table :admin_obras
    Admin::Obra.drop_translation_table!
  end
end
```

Código 5.3: Migración de la tabla "admin_obras"

```
==  CreateAdminArtista: migrating ============================================
-- create_table(:admin_artista)
   -> 0.0351s
==  CreateAdminArtista: migrated (0.2940s) ===================================

==  CreateAdminColeccions: migrating =========================================
-- create_table(:admin_coleccions)
   -> 0.0344s
==  CreateAdminColeccions: migrated (0.2407s) ================================

==  CreateObras: migrating ===================================================
-- create_table(:admin_obras)
   -> 0.0906s
==  CreateObras: migrated (0.3412s) ==========================================

==  AddObraColumnImagen: migrating ===========================================
-- add_column(:admin_obras, :imagen_file_name, :string)
   -> 0.1174s
-- add_column(:admin_obras, :imagen_content_type, :string)
   -> 0.1615s
```

Figura 5.2: Ejemplo de ejecución de *rake migrate*

Utilizar migraciones presenta otra gran ventaja y es que permite llevar un control de versiones de cambios del schema de la base de datos, permitiendo volver atrás tras una modificación errónea.

Para que la estructura sea migrada a la BD se utiliza la siguiente instrucción:

$>rake db:migrate

y, por medio de ella, *Rails* explora el contenido del directorio *db\migrate* y ejecuta las instrucciones contenidas en cada uno de los archivos *(figura 5.2)*.

Rails, además de las tablas definidas en las migraciones, añade otra tabla llamada **schema_migrations** en la que almacena la versión actual de las migraciones. De este modo, *Active Record* puede seguir el rastro de los cambios realizados en el esquema de cada tabla.

5.3.1.2. Manipulación de Datos

Para la *manipulación de datos* ***Active Record*** [BuenasTareas.com] provee a la aplicación de un fichero del objeto *modelo*. Éste podemos localizarlo en el directorio *"app\models"* con el nombre:

<nombre del modelo>.rb

Para el desarrollo de las clases del modelo he realizado los siguientes pasos en el fichero anterior:

- definir las relaciones entre los diferentes modelos,

- definir las validaciones oportunas y
- programar la lógica de negocio de la aplicación que recaía en cada clase.

Como ejemplo, muestro el modelo para la clase "*Admin::Obra*".

Lo primero que podemos observar es que la clase hereda de la clase ActiveRecord::Base, con lo que el programa averigua automáticamente qué tabla debe usar y qué columnas tiene.

```
class Admin::Obra < ActiveRecord::Base
```

Código 5.4: Cabecera del modelo "Admin::Obra"

Las relaciones en *Rails* se expresan mediante sencillas e intuitivas instrucciones como **has_one, belongs_to, has_many**. Veamos como se expresaría una de estas relaciones en *Rails* para la aplicación.

Relación (1:N) - "Artista crea Obra"

Esta relación expresa que una *obra* puede ser creada por un *artista* y que un *artista* puede crear n *obras*. En código *Rails* se expresa de la siguiente forma.

Dentro de la carpeta *app\models* en la:

- clase *Admin::Artistum*, deberá añadirse:

```
class Admin::Artistum < ActiveRecord::Base
  has_many :admin_obras, :class_name => "Obra"  ...
end
```

Código 5.5: Modelo "Admin::Artistum"

- clase *Admin::Obra* habrá que añadir:

```
class Admin::Obra < ActiveRecord::Base
  belongs_to :admin_artistum,  :class_name => "Artistum"
end
```

Código 5.6: Modelo "Admin::Obra"

Para el modelo de datos del ejemplo (Código 5.7) existe una serie de relaciones "belongs_to" traducidas a que la obra pertenece a un artista, a una colección, a una sección y a una técnica. También podemos ver la existencia de relaciones "has_many" traducidas a que pertenece a uno o varios carritos a través de la clase CarritoItems y a uno o varios pedidos a través de la clase PedidoItems.

```
class Admin::Obra < ActiveRecord::Base
  ...

  ### Relaciones ###
  # Relacion de uno a uno
  belongs_to :admin_artistum,   :class_name => "Artistum"
  belongs_to :admin_coleccion,  :class_name => "Coleccion"
  belongs_to :admin_seccion,    :class_name => "Seccion"
  belongs_to :admin_tecnica,    :class_name => "Tecnica"

  # Relacion de uno a muchos
  has_many :carritos, :through => :carrito_items
  has_many :carrito_items, :foreign_key => "admin_obra_id"

  has_many :pedido_items, :foreign_key => "admin_obra_id"
  has_many :pedidos, :through => :pedido_items

  ...
end
```

Código 5.7: Código de las relaciones del modelo "Obra"

También se puede contemplar una serie de validaciones que aseguran que los datos que se guardan en el modelo de datos son correctos.

```
class Admin::Obra < ActiveRecord::Base
  ...

  ### Validacion de datos ###
  # Restringimos los strings a 255 caracteres
  validates_length_of :titulo,      :in => 1..255
  validates_length_of :medidas,     :in => 1..255
  validates_length_of :referencia,  :in => 1..255

  # Ningun campo podra estar en blanco o vacio
  validates_presence_of :titulo
  validates_presence_of :referencia
```

```
    validates_presence_of :anio_realizacion
    validates_presence_of :medidas
    validates_presence_of :precio
    validates_presence_of :admin_artistum
    validates_presence_of :admin_seccion

    # Asociacion
    validates_associated :admin_artistum, :admin_coleccion,
      :admin_seccion, :admin_tecnica

    # Debe ser un numero positivo
    validates :precio,
      :numericality => {:greater_than_or_equal_to => 0.01}

    # Valor unico
    validates_uniqueness_of :referencia

    # Paperclip
    validates_attachment_presence :imagen, :message => :blank
    validates_attachment_content_type :imagen,
      :content_type=>['image/jpeg', 'image/png', 'image/gif']
    validates_attachment_size :imagen,
      :greater_than => 1.kilobyte

    ...
end
```

Código 5.8: Código de las validaciones del modelo "Obra"

Por último tenemos la lógica de negocio de la entidad.

```
class Admin::Obra < ActiveRecord::Base
  ...

  # Obras mas recientes
  def self.latest
    find :all, :limit => 12, :order => "admin_obras.id desc",
      include => [:admin_artistum, :admin_coleccion,
                  :admin_seccion]
  end

  # Metodo para sunspot
  searchable do
```

```
      text :titulo,            :default_boost => 6
      text :alias_artista,     :default_boost => 5
      text :nombre_artista,    :default_boost => 4
      text :nombre_seccion,    :default_boost => 3
      text :anio_realizacion,  :default_boost => 2
      text :fecha_publicacion
    end

    # Alias del artista
    def alias_artista
      return admin_artistum.i_alias
    end

    # Nombre del artista
    def nombre_artista
      return admin_artistum.nombre
    end

    # Nombre de la seccion
    def nombre_seccion
       admin_seccion.nombre
    end
  end
```

Código 5.9: Código de los métodos de la clase

5.3.1.3. Configuración de la base de datos

Dentro del directorio *"config"* se encuentra el fichero *"database.yml"* en el que se definen las BBDD que utilizará el sistema, así como el nombre de usuario y contraseña para su acceso.

He definido tres bases de datos, una para cada uno de los entornos definidos en el directorio *"config\environments"*: *development, test, production* (descritos en la sección 5.2).

```
development:
  adapter: mysql
  database: galeriaArte_development
  username: galeriaArte
  password: hacked
  encoding: utf8
```

```
test:
  adapter:  mysql
  database: galeriaArte_test
  username: galeriaArte
  password: hacked
  encoding: utf8

production:
  adapter:  mysql
  database: galeriaArte_production
  username: galeriaArte
  password: hacked
  encoding: utf8
```

Código 5.10: Contenido del fichero *"database.yml"*

Como comentaré anteriormente en la *Sección 5.3.1.1 "Definición de datos"*, para definir la información que se desea almacenar, *Rails* emplea unos ficheros de migración abstrayéndose de que tipo de BD se va a utilizar. De esta forma se podrá utilizar la aplicación con otras BBDD, sin tener que volver a definir las variables que se van a almacenar. Para ello bastará con migrar[4] la información de los ficheros a la nueva base de datos.

Prácticamente se ha abstraído la existencia de una base de datos durante la programación en *Rails*. El almacenamiento o eliminación de los datos han sido gestionados por medio de los *modelos*, nunca mediante comandos propios de la base de datos (aunque se podrían haber usado). Las BBDD han sido gestionadas mediante la herramienta *rake* a través de órdenes como:

```
$>rake db:create
$>rake db:drop
$>rake db:fixtures:load
```

5.3.2. Implementación del Controlador

Como vimos anteriormente en la *Sección 4.1.2 "Arquitectura MVC en Rails"* de la *página 104*, las clases del **Controlador** responden a la interacción del usuario e invocan a la *lógica de la aplicación*, que a su vez manipula los datos de las clases del *Modelo* y muestra los resultados usando las *Vistas*.

[4]Solamente se trasladan las definiciones.

En las aplicaciones web basadas en *MVC*, los métodos del controlador son invocados por el usuario a través de las peticiones enviadas desde el navegador web.

He implementado en el directorio *"app\controllers"* varios controladores mediante clases con el nombre:

<nombre de la tabla>_controller.rb

para gestionar las diferentes acciones de mi aplicación y también, he definido el controlador común de toda la aplicación, **Application Controller**, del que heredan todas las clases.

```
class Admin::ObrasController < ApplicationController
```

Código 5.11: Cabecera del controlador "Admin::Obras".

En el desarrollo de los controladores podemos distinguir los siguientes pasos:

- desarrollar las acciones e
- implementar el enrutamiento de las peticiones web a los métodos del controlador.

5.3.2.1. Desarrollar las acciones del controlador

Las acciones correspondientes a un modelo son gestionadas por su controlador. La implementación del método correspondiente a la acción del controlador crea las instancias necesarias para pasar a las *Vistas* de la aplicación o tan solo redirige hacia otras acciones. La comunicación se realiza a través de variables de instancia que son compartidas por las *Vistas* y los *Controladores*.

A continuación podemos observar, como ejemplo, la implementación de uno de los controladores de la aplicación.

Lo primero que he implementado en los controladores es el tipo de codificación que debe admitir. ***UTF-8*** *es capaz de representar cualquier carácter en el estándar Unicode.*

```
# encoding: utf-8
```

Código 5.12: Implementación del controlador "Admin::Obras". UTF-8

Seguidamente vemos las definiciones de los helpers y los filtros del controlador.

```
# encoding: utf-8

class Admin::ObrasController < ApplicationController
  helper_method :sort_column, :sort_direction

  before_filter :set_obra,
    :only => [:show, :edit, :edit_traduccion,
              :update, :destroy]
  ....

  private        # Uso interno por el controlador
    def load_data
      @admin_artistas    = Admin::Artistum.find(:all)
      @admin_coleccions  = Admin::Coleccion.find(:all)
      @admin_seccions    = Admin::Seccion.order("nombre ASC")
      @admin_tecnicas    = Admin::Tecnica.order("nombre ASC")
      @tags = Tag.find(:all)
    end

    def sort_column
      Admin::Obra.column_names.
           include?(params[:sort]) ? params[:sort] : "titulo"
    end

    def sort_direction
      %w[asc desc].include?(params[:direction])
            ? params[:direction] : "asc"
    end
    ...
end
```

Código 5.13: Implementación del controlador "Admin::Obras". Helpers y filtros

El siguiente código muestra un ejemplo de alguno de los métodos definidos en el controlador para implementar las diferentes historias de usuario.

```
class Admin::ObrasController < ApplicationController
  ....
  # GET /admin/obras
  def index
    # Paginacion con la gema 'will_paginate' y ordenacion
    @admin_obras = Admin::Obra.
          order(sort_column + ' ' + sort_direction).paginate
```

```
            :page => params[:page], :per_page => 5)

    store_location

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @admin_obras }
    end
  end

  # GET /admin/obras/1
  def show
    store_location

    respond_to do |format|
      format.html # show.html.erb
      format.xml  { render :xml => @admin_obra }
    end
  end

  # GET /admin/obras/new
  def new
    load_data
    @admin_obra = Admin::Obra.new
    @page_title = 'Nueva Obra'
    @action     = 'Crear Obra'

    respond_to do |format|
      format.html # new.html.erb
      format.xml  { render :xml => @admin_obra }
    end
  end ....
end
```

Código 5.14: Implementación del controlador "Admin::Obras". Métodos

5.3.2.2. Enrutamiento

El enlace entre las acciones[5] y las peticiones del navegador se encuentran definidas en el archivo *routes.rb* del directorio *"config"*.

[5] *Acciones:* Métodos de los controladores

```
GaleriaArte::Application.routes.draw do
  ....
  # Obras
  namespace :admin do
    resources :obras end
    post "obras/update_tecnicas" => "obras#update_tecnicas"
  end

  # Internacionalizacion espanol
  scope '(/:locale)', :locale => /es/ do
    namespace :admin do resources :coleccions end
    namespace :admin do resources :artista  end
    namespace :admin do resources :seccions end
    namespace :admin do resources :tecnicas end
  end  ....
end
```

Código 5.15: Extracto del contenido del fichero "routes.rb"

Por ejemplo, la siguiente línea de código anterior

```
    namespace :admin do
      resources :obras end
```

Código 5.16: Línea de código para el enrutamiento de "obras"

nos habilita automáticamente las siguientes URL´s para el controlador "admin/obras":

Verbo	**URL**	**Acción**	**Usado para**
GET	/admin/obras	index	Muestra un listado de todas las obras.
GET	/admin/obras/new	new	Devuelve un formulario para crear una obra.
POST	/admin/obras	create	Crea una nueva obra.
GET	/admin/obras/1/edit	edit	Devuelve un formulario para editar una obra.
GET	/admin/obras/1	show	Muestra una determinada obra.
PUT	/admin/obras/1	update	Actualiza una determinada obra.
DELETE	/admin/obras/1	destroy	Borra una obra determinada.

Tabla 5.1: URL´s habilitadas del enrutamiento de "obras"

Un listado de todas las rutas disponibles en la aplicación se obtiene a través de la orden:

```
$>rake routes
```

```
                          GET    (/:locale)/admin/obras/page/:page/:direction/:sort(.:format) {:controller=>"admin/obras", :action=>"index", :locale=>/es|en/}
                          GET    (/:locale)/admin/obras/page/:page(.:format)                  {:controller=>"admin/obras", :action=>"index", :locale=>/es|en/}
                          GET    (/:locale)/admin/obras/ordena/:direction/:sort(.:format)     {:controller=>"admin/obras", :action=>"index", :locale=>/es|en/}
                admin_obras GET    (/:locale)/admin/obras(.:format)                             {:controller=>"admin/obras", :action=>"index", :locale=>/es|en/}
                          POST   (/:locale)/admin/obras(.:format)                             {:controller=>"admin/obras", :action=>"create", :locale=>/es|en/}
            new_admin_obra GET    (/:locale)/admin/obras/new(.:format)                         {:controller=>"admin/obras", :action=>"new", :locale=>/es|en/}
           edit_admin_obra GET    (/:locale)/admin/obras/:id/edit(.:format)                    {:controller=>"admin/obras", :action=>"edit", :locale=>/es|en/}
                admin_obra GET    (/:locale)/admin/obras/:id(.:format)                         {:controller=>"admin/obras", :action=>"show", :locale=>/es|en/}
                          PUT    (/:locale)/admin/obras/:id(.:format)                         {:controller=>"admin/obras", :action=>"update", :locale=>/es|en/}
                          DELETE (/:locale)/admin/obras/:id(.:format)                         {:controller=>"admin/obras", :action=>"destroy", :locale=>/es|en/}
 admin_obras_update_tecnicas POST   (/:locale)/admin/obras/update_tecnicas(.:format)          {:controller=>"admin/obras", :action=>"update_tecnicas", :locale=>/es
en/}
          admin_coleccions GET    (/:locale)/admin/coleccions(.:format)                        {:controller=>"admin/coleccions", :action=>"index", :locale=>/es/}
                          POST   (/:locale)/admin/coleccions(.:format)                        {:controller=>"admin/coleccions", :action=>"create", :locale=>/es/}
        new_admin_coleccion GET    (/:locale)/admin/coleccions/new(.:format)                    {:controller=>"admin/coleccions", :action=>"new", :locale=>/es/}
       edit_admin_coleccion GET    (/:locale)/admin/coleccions/:id/edit(.:format)               {:controller=>"admin/coleccions", :action=>"edit", :locale=>/es/}
            admin_coleccion GET    (/:locale)/admin/coleccions/:id(.:format)                    {:controller=>"admin/coleccions", :action=>"show", :locale=>/es/}
                          PUT    (/:locale)/admin/coleccions/:id(.:format)                    {:controller=>"admin/coleccions", :action=>"update", :locale=>/es/}
                          DELETE (/:locale)/admin/coleccions/:id(.:format)                    {:controller=>"admin/coleccions", :action=>"destroy", :locale=>/es/}
             admin_artista GET    (/:locale)/admin/artista(.:format)                           {:controller=>"admin/artista", :action=>"index", :locale=>/es/}
                          POST   (/:locale)/admin/artista(.:format)                           {:controller=>"admin/artista", :action=>"create", :locale=>/es/}
          new_admin_artistum GET    (/:locale)/admin/artista/new(.:format)                       {:controller=>"admin/artista", :action=>"new", :locale=>/es/}
         edit_admin_artistum GET    (/:locale)/admin/artista/:id/edit(.:format)                  {:controller=>"admin/artista", :action=>"edit", :locale=>/es/}
              admin_artistum GET    (/:locale)/admin/artista/:id(.:format)                       {:controller=>"admin/artista", :action=>"show", :locale=>/es/}
                          PUT    (/:locale)/admin/artista/:id(.:format)                       {:controller=>"admin/artista", :action=>"update", :locale=>/es/}
                          DELETE (/:locale)/admin/artista/:id(.:format)                       {:controller=>"admin/artista", :action=>"destroy", :locale=>/es/}
```

Figura 5.3: Extracto del listado de rutas de la aplicación

5.3.3. Implementación de la Vista

La **Vista** es el objeto que maneja la presentación visual de los datos representados por el *Modelo*. Genera una representación visual del *Modelo* y muestra los datos al usuario.

La interacción con el usuario en una aplicación web se realiza mediante un navegador que interpreta las páginas *HTML* enviadas desde el servidor.

Aunque existen muchas maneras de gestionar las *Vistas*, el método por defecto que se emplea en *Rails* es ***Ruby Empotrado***, que consiste en fragmentos de código *HTML* con algo de código en *Ruby*, generando así *HTML dinámico*. Podemos diferenciar el código *Ruby* ya que se representa mediante el uso de:

- **Scriptlets**. Código situado entre las etiquetas $<\%\ \%>$.
- **Expresiones**. Código situado entre las etiquetas $<\%=\ \%>$.

Un ejemplo podemos verlo en los *Códigos 5.17 y 5.18* de la *página 155*.

Las vistas del proyecto las podemos localizar dentro del directorio *"app\views"*, donde cada controlador tiene su propio directorio en el que se guardan todas las plantillas *RHTML* con la siguiente estructura:

```
<nombre del controlador>\<accion>.html.erb
```

5.3.3.1. Layout

RoR permite utilizar una plantilla común para toda la aplicación llamada *"layout"* donde, según la petición recibida, se incrusta la plantilla de la vista solicitada. Esta, que será la plantilla principal de mi aplicación, modela los elementos comunes de la aplicación como: el encabezado, el menú principal, el pie de página...

Podemos localizarla también dentro del directorio *"app\views"* con la siguiente estructura:

\layouts\application.html.erb

5.3.3.2. Partials

Siguiendo uno de los principios de la filosofía de *Rails "Don't Repeat Yourself"* descrito en el *Párrafo "Filosofía"* de la *página 268*, he trabajado con los llamados ***"partials"*** que son fragmento de código *"html.erb"* que pueden ser insertados en cualquier *vista*. La ventaja de su uso es eliminar la duplicación innecesaria de código.

Los *partials* implementados para la aplicación podemos encontrarlos dentro de cada uno de los directorios que componen las *vistas* con la siguiente estructura:

<_nombre del partial>.html.erb

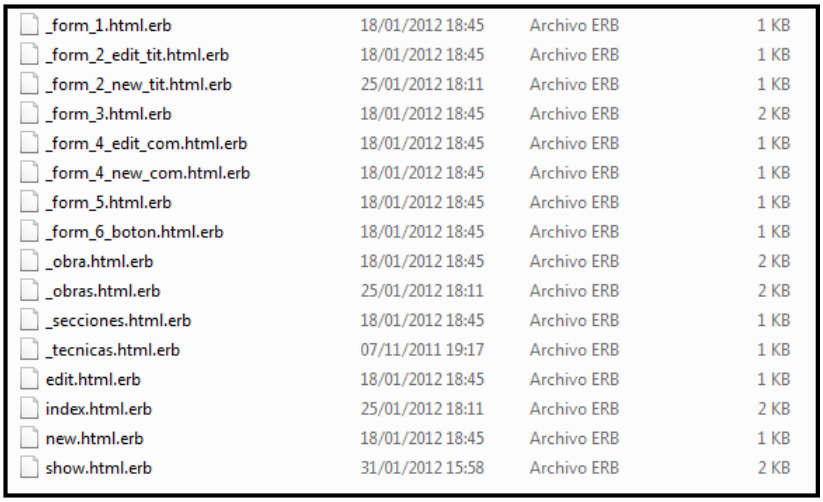


Figura 5.4: Ejemplo de los partials de las Vistas de Obras

A continuación, a modo de ejemplo, podemos ver el código de la vista "new.html.erb" en donde se introduce el uso de varios *partials* para implementar el formulario. Podemos observar que se consigue un código limpio y fácil de seguir.

```
<div id="page_content">
  <%= form_for(@admin_obra, @admin_artistas,
     @admin_coleccions, @tags,
     :html => {:multipart => true}) do |f| %>
    <%= render 'form_1', :f => f %>
    <%= render 'form_2_new_tit', :f => f %>
    <%= render 'form_3', :f => f %>
    <%= render 'form_4_new_com', :f => f %>
    <%= render 'form_5', :f => f %>
    <%= render 'form_6_boton', :f => f  %>
  <% end %>

  <%= link_to '', admin_obras_path, :class => "back",
     :title => "Retroceder"%>
</div>
```

Código 5.17: Implementación de la vista "new.html.erb" utilizando partial

En el siguiente código podemos ver la implementación del primero de los *partials* a los que se hace referencia en el código anterior.

```
  <% if @admin_obra.errors.any? %>
    <div id="error_explanation">
      <h2>
        <%= t('errors.template.header',
             :count=>@admin_obra.errors.size,
             :model=>t('activerecord.models.admin_obra')) %>:
      </h2>

      <ul>
      <% @admin_obra.errors.full_messages.each do |msg| %>
        <li><%= raw msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>
```

Código 5.18: Implementación del partial "_form_1.html.erb"

5.3.3.3. Hojas de estilo

Para realizar la maquetación de la aplicación he usado las ***Hojas de Estilo en Cascada (CSS)*** las cuales son básicas para que los *HTML* se vean como deseamos. Éstas son muy importantes en el proceso de la creación de un sitio Web, por eso he procurado que sean sencillas, entendibles y, sobre todo, que estén bien estructuradas de forma que sean fácilmente modificables.

Podemos localizarla dentro del directorio *"public\stylesheets"* con la siguiente estructura:

\nombre_hoja_estilo.css

```
/* ------- Catalogo ---------- */

/* Botones en catalogo */
a.img_recent {
  text-decoration:none;
  display:inline-block;
  width:22px;
  height:22px;
  background-image: url('../images/last_art.png');
}

a.img_recent:hover {
  display:inline-block;
  width:22px;
  height:22px;
  background-image: url("../images/last_art-over.png");
  background-color: transparent;
}

a.img_cesta {
  text-decoration:none;
  display:inline-block;
  width:22px;
  height:22px;
  background-image: url('../images/cart_24.png');
}

a.img_cesta:hover {
  display:inline-block;
  width:22px;
  height:22px;
```

```
    background-image: url("../images/cart_24-over.png");
    background-color: transparent;
}
...
```

Código 5.19: Extracto de hoja de estilo (catalogo.css)

5.3.4. Sesiones y Autenticación

El sistema de sesiones y autenticación de la aplicación se ha delegado en la librería de *Ruby on Rails* **Authlogic** [Johnson, Ben; ASCIIcasts]. Se caracteriza por:

- Ser muy fácil de configurar, utilizar y extender.
- Tiene muy buen nivel de seguridad y muchas funcionalidades extremadamente útiles que, de hacerlas a mano, tomaría mucho tiempo de implementar.
- No genera controladores o vistas sino que sólo trabaja con el código que gestiona la autenticación de los usuarios. Debido a esto lleva un poco más de tiempo hacer que la aplicación funcione con *Authlogic* ya que hay que escribir un poco más de código, pero se gana flexibilidad y control acerca de cómo se presenta la autenticación al usuario.

En la implementación del sistema de autenticación he creado la clase del modelo *User*, que gestiona la información básica necesaria para el registro de los usuarios (nombre de usuario y contraseña). Este modelo incluye `acts_as_authentic` para activar la funcionalidad de *authlogic*.

He creado también el modelo *UserSession*, el cual se encarga de administrar todo lo que tiene que ver con una sesión en nuestro sitio: su creación (login) y su destrucción (logout).

```
class UserSession < Authlogic::Session::Base
```

Código 5.20: Cabecera del modelo *UserSession* (*"models\user_session.rb"*)

Hay que destacar que este modelo hereda de *Authlogic::Session::Base*, lo cual permite configurarlo fácilmente.

Authlogic se encarga simplemente de la autenticación de usuarios (el encriptamiento de la contraseña lo realiza de forma automática), pero permite restringirles el acceso en los controladores. Para ello he definido los siguientes métodos:

```
class ApplicationController < ActionController::Base
  def require_usuario
    unless current_user
      store_location
      redirect_to(new_user_session_url,
          :notice => "Usted debe estar conectado
          para tener acceso")
      return false
    end
  end

  def require_no_usuario
    if current_user
      store_location
      redirect_to(catalogo_path,
          :notice => "Usted debe cerrar su sesion
          para acceder a esta pagina")
      return false
    end
  end  ...
```

Código 5.21: Restricción de acceso (*"controllers\application_controller.rb"*)

los cuales utilizo a través de filtros en los controladores. Por ejemplo, solo los usuarios registrados podrán realizar compras de obras de arte.

```
# app\controllers\checkout_controller.rb
class CheckoutController < ApplicationController
  ...

  # Se requiere ser usuario registrado
  before_filter :require_usuario
  ...
end
```

Código 5.22: Restricción de acceso a la cesta de la compra

De esta forma, si un usuario visitante intenta realizar una compra, será redirigido automáticamente al formulario de "Inicio de sesión" para que se se registre o se identifique con su usuario antes de poder continuar.

5.3.5. Búsqueda de obras

En la aplicación se ha implementado la opción de que se puedan realizar búsquedas de obras de arte.

Esta parte me ha llevado bastante tiempo implementarla, ya que al principio me decidí hacerlo utilizando la gema "Sunspot" [Brown, Mat], que permite añadir búsqueda de texto completo en la aplicación y, además, tiene muchas características interesantes. Ésta me dio bastantes quebraderos de cabeza y una vez que conseguí que funcionara, cuando pase a probar la aplicación en Heroku, me encontré con la sorpresa de que el manejo de esa gema en el servidor costaba dinero para los desarrolladores. Así que tuve que buscar una nueva solución para implementarla, la gema "MetaSearch" [Miller; Railscasts].

Esta gema añade una forma muy útil de realizar búsquedas sobre un modelo desde un formulario, que consiste en llamar a un método llamado "search" sobre cualquier modelo pasándole los parámetros de búsqueda, tras lo que se recuperan los registros que casan con dicha consulta.

```
# controllers\admin\obras_controller.rb
class Admin::ObrasController < ApplicationController
  ...
  def index
    ...
    # Busqueda con la gema 'meta_search'
    @search = Admin::Obra.search(params[:search])
    @admin_obras =
          @search.order(sort_column + ' ' + sort_direction).
          paginate(:page => params[:page], :per_page => 5)
    ...
  end
  ...
end
```

Código 5.23: Llamada al método *Search* con gema "MetaSearch"

En el ejemplo anterior de la aplicación, he creado la búsqueda invocando a "Admin::Obra.search", pasándole los parámetros del formulario. A continuación se invoca a @search.order para recuperar la lista de productos correspondiente de forma ordenada y paginada.

En el código de la vista, el nombre de cada campo del formulario determinan las condiciones de búsqueda.

```
<div class="field">
  <%= f.label :admin_artistum_i_alias_contains,
      t('.artista') %><br />
  <%= f.text_field :admin_artistum_i_alias_contains %>
      <br /><br />
</div>

<div class="field">
  <%= f.label :titulo_contains, t('.titulo') %><br />
  <%= f.text_field :titulo_contains %><br /><br />
</div>
...
```

Código 5.24: Nombre de los campos para la búsqueda con gema "MetaSearch"

En el ejemplo anterior, el campo de texto llamado "titulo_contains" quiere decir que se buscarán los registros cuyo titulo contenga el valor introducido por el usuario en dicho campo.

5.3.6. Notificaciones por correo electrónico

En la instalación por defecto de *Rails* existe una librería llamada *ActionMailer* [Lindsaar, Mikel; ASCIIcasts], cuya función es la de implementar el envío de correos electrónicos de una manera sencilla *(figura 5.5)*.

Sabiendo esto, crear un mailer para la aplicación ha resultado bastante sencillo, a través de la siguiente orden en la línea de comandos, dentro del directorio de la aplicación:

```
$>rails g mailer <clase Mailer>
```

En esta instrucción, la *clase Mailer* será una clase que se creará en el directorio *"app\mailers"*, que heredará de la clase principal de la librería *ActionMailer*. Cada una de las notificaciones que sea implementada dentro de esta clase, tendrá su correspondiente método dentro de la misma.

También se crea mediante la instrucción anterior, dentro del directorio *"app\views"*, una carpeta con el mismo nombre de la clase, que contendrá una vista por cada uno de los métodos del mail creados, donde se codificarán las distintas plantillas de las diferentes notificaciones que se deseen enviar por correo electrónico.

La aplicación web que he desarrollado también se encarga de enviar notificaciones por correo electrónico (figura 5.5), al cliente y/o a la galería, concretamente cuando un usuario:

Estimad@ cliente **"Silvia"**:

Ha solicitado la/s siguiente/s obra/s de nuestra Galeria de Arte 'GadeirArt':

Información del pedido

Ref. pedido: 3
Fecha: 27/2/2013

Ref.	Titulo	Uds.	Precio	Total
pint_105	Corazon...	1	160,00 €	160,00 €
fot_505	Cota de...	1	120,00 €	120,00 €

Base imponible:	231,40 €
IVA (21 %):	48,60 €
Subtotal:	280,00 €

Total: 280,00 €

Le enviaremos un correo electrónico cuando su pedido sea enviado a la dirección que nos ha facilitado.

Información de contacto

Email: akela_sil@yahoo.es
Teléfono: 956000000

Dirección de envío

Nombre: Silvia Garbarino de la Rosa
Dirección: Mi casita
Localidad: San Fernando
Provincia: Cádiz
CP: 11100
País: ES

Muchas gracias por su confianza.

Figura 5.5: Ejemplo de email de confirmación de pedido al cliente

- se registra en la web,
- no recuerda su contraseña y
- cuando realiza un pedido a la Galería de Arte.

Configuración del correo

Para la aplicación he especificado una configuración SMTP durante la inicialización.

Las opciones de dicha configuración podemos encontrarla en el fichero *"setup_mail.rb"* del directorio *"config\initializers"*.

```
# Establece el metodo de entrega
ActionMailer::Base.delivery_method = :smtp

# Opciones para la entrega smtp
ActionMailer::Base.smtp_settings = {
  :enable_starttls_auto => true,
  :address              => "smtp.gmail.com",
  :port                 => "587",
  :user_name            => "gadeirart@gmail.com",
  :password             => <password>,
  :authentication       => "plain",
  :enable_starttls_auto => true
}

# Despliegue
ActionMailer::Base.default_url_options[:host] =
  "gadeirart.heroku.com"

ActionMailer::Base.default :charset      => "utf-8"
ActionMailer::Base.default :content_type => "text/html"
```

Código 5.25: Configuración del servidor de correos (*"setup_mail.rb"*)

Implementación

La clase *mailer* se comportan de manera parecida a los controladores normales de la aplicación por lo que comparten gran parte del código.

Por cada notificación de correo que va a enviar la aplicación he añadido un método a esta clase. Veamos, como ejemplo, el método para la confirmación del registro de usuarios en la web de la galería:

```
class UserMailer < ActionMailer::Base
  # Los emails siempre se envian desde la misma direccion
  default :from => "gadeirart@gmail.com"

  # Confirmacion de registro
  def confirmacion_registro(user)
    @usuario_registrado = user

    mail(:to      => user.email,
         :subject => I18n.t('.notificacion_registro'))
          do |format|
              format.text # confirmacion_registro.text.erb
              format.html # confirmacion_registro.html.erb
          end
  end
  ...
end
```

Código 5.26: Metodo de confirmación de registro (*"user_mailer.rb"*)

Lo que hace el método anterior es invocar el método "mail" pasándole un hash con argumentos tales como: :to, :from, y :subject.

Los mailers de correo, al igual que los controladores, tienen asociado un archivo de vista, en el que se coloca el cuerpo del correo que se desea enviar. Para la aplicación he implementado correos en:

- texto plano: *<nombre del método del correo>.text.erb*
- HTML: *<nombre del método del correo>.html.erb*

Los emails se envían cuando se esta interactuando directamente con la propia aplicación, en cuyo caso se está pasando por el controlador. Continuando con el ejemplo anterior, vemos como se invoca al método desde el controlador.

```
class UsersController < ApplicationController
  ...
  def create
    ...
      # Envia un email de confirmacion de registro
      UserMailer.confirmacion_registro(@user).deliver
    ...
  end
  ...
end
```

Código 5.27: Controlador invocando al método del correo (*"user_controller.rb"*)

5.3.7. Pasarela de pago

La *Galería de Arte* desea un sistema de procesamiento, para los pedidos de sus clientes, a través de una pasarela de pago.

Este sistema se ha delegado en dos librería de *Ruby on Rails*:

- **Active Merchant** [Luetke, Tobias; Fauser, Cody]: Nos permite el acceso de forma sencilla a la pasarela de pago, haciendo frente a las tarjetas de crédito.
- **Authorize-Net** [Center]: Nos facilita la conexión a la pasarela de pago *Authorize.Net*, la cual proporciona la infraestructura compleja y de seguridad necesaria para garantizar transacciones seguras, rápidas y fiables.

He seleccionado *Authorize.Net*, como pasarela de pago debido a que no he encontrado ninguna española que tuviera facilidad para su implementación, ya que para ello necesitaba datos reales y recordamos que la Galería de Arte es ficticia.

Authorize.Net dispone de un apartado *sandbox (caja de arena)* donde poder realizar pruebas durante el desarrollo de la pasarela de pago. Para ello tuve que crearme una cuenta ficticia, la cual me permite ver que el pago a través de la pasarela funciona correctamente.

A continuación veremos el código a través del cual conectamos con la pasarela de pagos *Authorize.Net*, en el modo de prueba.

```
GaleriaArte::Application.configure do
  ...
  # Pasarela de pagos
  config.after_initialize do
    # Envia una solicitud al servidor de prueba
    # de la pasarela de pagos
    ActiveMerchant::Billing::Base.mode = :test

    # Crea un objeto pasarela al servicio AuthorizeNetGateway
    ::GATEWAY =
        ActiveMerchant::Billing::AuthorizeNetGateway.new(
         # API Credential AuthorizeNetGateway
         :login     => '8gNzqAW9S8d',
         :password  => '962ZT2Grx7h9jv5C'
    )
  end
end
```

Código 5.28: Conexión con *Authorize.net* (*"config\environments\development.rb"*)

En la imagen 5.28 podemos ver los datos de la API Credentials *facilitados al crear la cuenta.*

El acceso a ésta se consigue a través de la gema *Active Merchant*. Para ello he implementado el siguiente procedimiento:

```
def process_with_active_merchant
    # Crea una tarjeta crédito para el comprador
    credit_card = ActiveMerchant::Billing::CreditCard.new(
        :first_name         => entregar_a_nombre,
        :last_name          => entregar_a_apellidos,
        :type               => tipo_tarjeta,
        :number             => numero_tarjeta,
        :month              => mes_caducidad,
        :year               => anio_caducidad,
        :verification_value => codigo_verificacion
    )

    if credit_card.valid?
      # Informacion del comprador para su pedido
      options = {
        # Información del pedido
        :order_id          => self.id,
```

```
            :description       => 'Obras de ... GadeirArt',

            # Información de facturación
            :billing_address   => {
                    :address1  => direccion_de_entrega,
                    :city      => poblacion_de_entrega,
                    :state     => ciudad_de_entrega,
                    :zip       => cp_entrega,
                    :country   => pais_entrega,
                    :phone     => telefono,
            },
            :email             => email,

            # Información de envío
            :shipping_address  => {
                    :first_name => entregar_a_nombre + " "
                                   + entregar_a_apellidos,
                    :address1  => direccion_de_entrega,
                    :city      => poblacion_de_entrega,
                    :state     => ciudad_de_entrega,
                    :zip       => cp_entrega,
                    :country   => pais_entrega
            }
        }

        # Autorización para una cantidad de dinero
        response =
                GATEWAY.purchase(total, credit_card, options)
    ...
end
```

Código 5.29: Proceso con *Active Merchant* (*"models\pedido.rb"*)

En el código 5.29 se crea una tarjeta de crédito a través de los datos facilitados por el cliente y se recoge la información relacionada con la facturación del pedido.

Filter by: ALL View

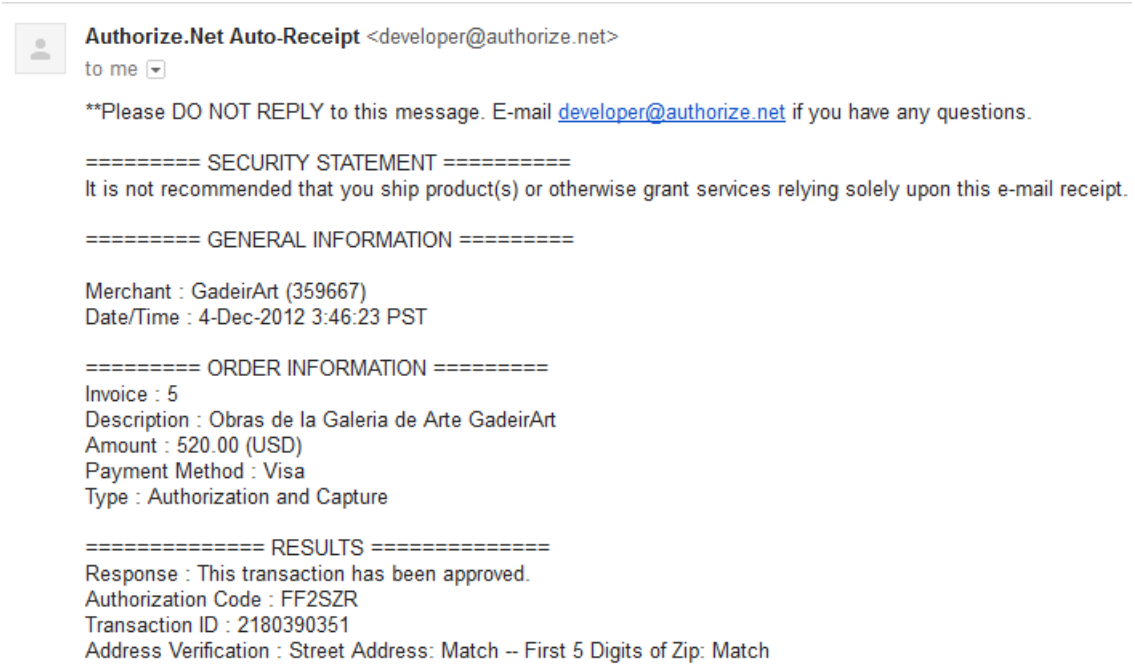
1-2 of 2 results

Trans ID	Invoice Number	Trans Status	Submit Date	Customer	Card	Payment Method	Payment Amount
2185045592	6	Captured/Pending Settlement	26-Feb-2013 06:48:03	Fernández, Javier	V	XXXX4242	USD 125.00
2185045527	5	Captured/Pending Settlement	26-Feb-2013 06:43:57	Rosa, Silvia	V	XXXX4242	USD 280.00

1-2 of 2 results

Figura 5.6: Ejemplo de transacciones recibidas por Authorized.net

Cuando un cliente realiza un pedido y decide pagarlo a través de su tarjeta de crédito, la pasarela de pagos *Authorize.Net* recibe la información sobre el pedido y autoriza su cobro *(figura 5.6)* informando de ello, a través de un email, a la galería. Podemos ver uno de esos email en la *figura 5.7*.



Authorize.Net Auto-Receipt <developer@authorize.net>
to me

**Please DO NOT REPLY to this message. E-mail developer@authorize.net if you have any questions.

========== SECURITY STATEMENT ==========
It is not recommended that you ship product(s) or otherwise grant services relying solely upon this e-mail receipt.

========== GENERAL INFORMATION ==========

Merchant : GadeirArt (359667)
Date/Time : 4-Dec-2012 3:46:23 PST

========== ORDER INFORMATION ==========
Invoice : 5
Description : Obras de la Galeria de Arte GadeirArt
Amount : 520.00 (USD)
Payment Method : Visa
Type : Authorization and Capture

============== RESULTS ==============
Response : This transaction has been approved.
Authorization Code : FF2SZR
Transaction ID : 2180390351
Address Verification : Street Address: Match -- First 5 Digits of Zip: Match

Figura 5.7: Correo enviado por Authorized.net con la aprobación del cobro

5.3.8. Internacionalización

Primeramente voy a definir un par de términos que creo importantes para esta sección.

Internacionalización proceso en el que se desarrolla una aplicación que puede adaptarse a distintos idiomas y regiones sin necesidad de realizar cambios de ingeniería.

Localización proceso de adaptar el software a una región o lenguaje específico añadiendo componentes específicamente locales y traduciendo el texto pertinente.

Recordemos que uno de los requisitos del sistema es que las páginas web de la tienda virtual puedan verse tanto en español como en inglés. Para la internacionalización/localización de la aplicación he utilizado dos gemas.

- **I18n:** Para traducir el texto estático de la web, guardando su traducción en ficheros *YAML* dentro de la aplicación.
- **Globalize3 [Stachewicz, Tomasz]:** Para mostrar el texto de la base de datos en diferentes idiomas.

I18n

Esta gema proporciona todos los medios necesarios para la internacionalización/localización y viene incluida en *Rails*, por lo que simplemente he tenido que configurar la aplicación para que utilice dicha funcionalidad. Los pasos seguidos han sido:

- Crear un fichero *yml* por cada uno de los idiomas que utiliza la aplicación en el directorio *"config\locales"*: español (*"es.yml"*) e inglés (*"en.yml"*) . Como ejemplo muestro el mismo extracto de cada uno de estos ficheros.

```
# Traduccion en espanol

"es":
  ...
  layouts:
      application:
        catalogo:           "Catalogo"
        etiquetas:          "Etiquetas"
        contactar:          "Contactar"
        quienes_somos:      "Quienes somos"
        condiciones:        "Condiciones"
        bienvenido:         "Bienvenid@"
        mi_cuenta:          "Mi cuenta"
        salir:              "Salir"
        registrarse:        "Registrarse"
        login:              "Iniciar sesion"
        es:                 "Espanol"
        en:                 "Ingles"
        mi_cesta:           "Mi compra"
  ...
```

Código 5.30: Fichero de traducción al español

```
# Traduccion en ingles

en:
  ...
  layouts:
      application:
        catalogo:           "Catalog"
        etiquetas:          "Tags"
        contactar:          "Contact us"
```

```
    quienes_somos:      "Who we are"
    condiciones:        "Terms"
    bienvenido:         "Welcome"
    mi_cuenta:          "My account"
    salir:              "Exit"
    registrarse:        "Register"
    login:              "Log-In"
    es:                 "Spanish"
    en:                 "English"
    mi_cesta:           "My shopping cart"
```

Código 5.31: Fichero de traducción al inglés

- Indicar a la aplicación que cargue los diferentes archivos de idioma, para lo he creado el archivo de inicialización *"config\initializers\i18n.rb"*.

```
# Decimos a la biblioteca I18n donde encontrar las
# traducciones para que las cargue
I18n.load_path +=
 Dir[Rails.root.join('lib', 'locale', '*.{rb,yml}')]
```

Código 5.32: Carga de diferentes archivos de idioma

- Seleccionar el idioma con el que deseo que se muestre la aplicación, es decir, español.

```
# Idioma por defecto
I18n.default_locale = :es
```

Código 5.33: Idioma por defecto

Con esto ha quedado configurada la aplicación para utilizar *I18n*. La traducción de las cadena de texto las he obtenido utilizando el helper *I18n.translate* a través de su alias t.

```
...
<body>
  ...
    <div id="top_banner">
      <div id="top_menu">
```

```
        <ul class="menu">
            ...
            <li><a href="/inicio" class="nav">
                    <%= t('.inicio') %>
            </a></li>
            <li><a href="/galeria/nosotros/1" class="nav">
                    <%= t('.quienes_somos') %>
            </a></li>
            <li><a href="/catalogo" class="nav">
                    <%= t('.catalogo') %>
            </a></li>
            <li><a href="/galeria/condiciones/index_usuario"
                   class="nav">
                    <%= t('.condiciones') %>
            </a></li>
            <li><a href="/contactar"  class="nav">
                    <%= t('.contactar') %>
            </a></li>
        </ul>
      </div>
    </div>
...
```

Código 5.34: Idioma por defecto

En el ejemplo anterior de la aplicación, t('.quienes_somos') nos devuelve "Quienes somos" para el idioma español y "Who we are" si la página se esta mostrando en inglés, según el correspondiente fichero de traducción.

Globalize3

Globalize3 también provee servicios para la internacionalización y localización en *Rails*.

Como comenté anteriormente, esta gema me ha ayudado a mostrar el texto de la base de datos en diferentes idiomas, para lo que crea una tabla independiente para cada uno de los modelos de la aplicación donde almacena las diferentes traducciones. Según la zona seleccionada por el usuario en la web pasará a mostrar la interpretación correspondiente.

```
class CreateObras < ActiveRecord::Migration
  def self.up
    ...
    # Traduccion
    Admin::Obra.create_translation_table!
                :titulo      => :string,
                :comentario  => :text,
                :estado      => :string
  end

  def self.down
    ...
    Admin::Obra.drop_translation_table!
  end
end
```

Código 5.35: Tabla para la traducción del modelo Obras

El código anterior crea una tabla para la traducción del modelo Obras con los campos que se desean traducir.

Capítulo 6

Pruebas del Sistema

Las *pruebas de software («testing»)* [Wikipedia] son los procesos que permiten *verificar* y *revelar la calidad* de un producto software. Son utilizadas para *identificar* posibles *fallos de implementación, calidad* o *usabilidad* de un programa. Básicamente es una fase en el desarrollo de software consistente en probar las aplicaciones construidas.

Para ello, seguiré un *desarrollo guiado por pruebas - TDD* con el que lograré un código limpio que funcione. Este consiste en dos prácticas:

- *Escribir las pruebas primero («Test First Development»)*, y se verifica que estas fallen, luego se implementa el código que haga que la prueba pase satisfactoriamente.
- *Refactorización («Refactoring»)* del código escrito.

La descripción de las diferentes pruebas que llevaré a cabo durante el desarrollo del proyecto se encuentra en el *anexo F "Pruebas"* y tendrán dos finalidades:

- Dar soporte y guiar la fase de desarrollo e implementación.
- Dar una garantía del correcto funcionamiento de las distintas funcionalidades implementadas, para poder así asegurar la calidad del producto final entregado.

6.1. Pruebas en Rails

Rails posee, en su árbol de directorios, una carpeta denominada **test** orientada a la implementación de las pruebas, la cual posee los siguientes subdirectorios:

- **fixtures:** Donde se incluirán una serie de ficheros *YAML*, en los cuales será posible crear, de forma sencilla e intuitiva, las instancias que serán empleadas en las distintas pruebas.

- **functional:** En esta carpeta se codificarán las pruebas encargadas de garantizar el correcto funcionamiento de los distintos controladores de la aplicación.
- **integration:** Se guardarán las pruebas realizadas para comprobar la interacción entre los diferentes controladores.
- **unit:** Se almacenarán las distintas pruebas unitarias que se lleven a cabo sobre los modelos de datos de los distintos módulos a probar.

La creación de los ficheros con las plantillas para poder programar los test, incluidos en los subdirectorios anteriores, se realiza de forma automática cuando se utilizan algunos de los scripts incluidos en *Rails*, como por ejemplo el **Scaffold** *(ver sección 5.2)*.

La aplicación web que voy a desarrollar está basada en el patrón de diseño *MVC («Modelo, Vista y Controlador») (ver sección 4.1.2)*, siendo cada uno de estos componentes independientes entre sí. Este hecho, trasladado a la fase de pruebas, lleva como consecuencia que algunas de las pruebas que se realicen se centren en aspectos cuya funcionalidad resida en el modelo, en el controlador o en la vista; y que según donde se establezca, las pruebas necesiten un enfoque o unas herramientas concretas.

6.2. Pruebas unitarias

Los modelos de datos incluye una serie de normas, excepciones y relaciones adaptados al problema a resolver. Para poder comprobar su correcto cumplimiento, he realizado una serie de *pruebas unitarias*, cuyo objetivo será el de garantizar el cumplimiento de las condiciones de validación tratando que ningún tipo de información que las vulnere sea almacenada en la base de datos. Podemos localizar todas estas pruebas en el directorio *"test\unit"* de la aplicación con la siguiente estructura de nombre:

<nombre del modelo>_test.rb

A continuación, a modo de ejemplo, muestro un extracto de las pruebas unitarias realizadas sobre el modelo *"Obras"*.

```
class Admin::ObraTest < ActiveSupport::TestCase
  ...

  ##########################################################
  # Test para comprobar que no puede guardarse #
  # una obra sin titulo                                    #
  ##########################################################
  test "Test no guardar obra sin titulo" do
    obra = Admin::Obra.new(
```

```
                #:titulo             => 'Gato pardo',
                :referencia          => '123',
                :anio_realizacion    => 2010,
                :medidas             => '20 x 20',
                :precio              => 35.50,
                :admin_coleccion_id => 1,
                :admin_artistum_id  => 1,
                :admin_seccion_id    => 1,
                :admin_tecnica_id    => 1)
  assert !obra.save, "Guardado obra sin titulo"
end

##################################################################
# Test para comprobar que el precio es positivo #
##################################################################
test "Test precio de la obra debe ser positivo" do
   obra = Admin::Obra.new(
                :titulo              => 'Gato pardo',
                :referencia          => '321',
                :anio_realizacion    => 2010,
                :medidas             => '10 x 15',
                :precio              => 35.50,
                :admin_coleccion_id  => 1,
                :admin_artistum_id   => 1,
                :admin_seccion_id    => 1,
                :admin_tecnica_id    => 1,
                :imagen_file_name    => 'cuadro.jpg',
                :imagen_content_type => 'image/jpeg',
                :imagen_file_size    => 34714)

   obra.precio = -1
   assert obra.invalid?
   assert_equal "debe ser mayor que o igual a 0.01",
                obra.errors[:precio].join('; ')

   obra.precio = 0
   assert obra.invalid?
   assert_equal "debe ser mayor que o igual a 0.01",
                obra.errors[:precio].join('; ')

   obra.precio = 35.50
   assert obra.valid?
end
...
```

```
##########################################################
# Test para verificar que se puede acceder a            #
# una conjunto de obras de un artista                    #
##########################################################
test "has_many_and_belongs_to_artista" do
  # Mira en los artistas y verifica que hay al menos
  # una obra en el conjunto de obras insertada por
  # fixture al comienzo del test.
  artista =
      Admin::Artistum.find_by_i_alias("Pedro González")
  assert_equal 2, artista.admin_obras.size

  # Creamos una nueva obra
  obra = Admin::Obra.new(
      :titulo              => 'Vista de la Catedral
                               de Cádiz',
      :referencia          => 'fot_511',
      :anio_realizacion    => 2011,
      :medidas             => '10 x 15',
      :precio              => 5.00,
      :admin_coleccion_id  => Admin::Coleccion.find(3).id,
      :admin_artistum_id   => Admin::Artistum.find(1).id,
      :admin_seccion_id    => Admin::Seccion.find(5).id,
      :admin_tecnica_id    => Admin::Tecnica.find(11).id,
      :imagen_file_name    => 'cadiz002.jpg',
      :imagen_content_type => 'image/jpeg',
      :imagen_file_size    => 29315)

  # Añadimos la nueva obra al artista
  artista.admin_obras << obra

  # Recargamos en la BD los datos
  artista.reload
  obra.reload

  # Verificamos que ahora hay dos obras para el artista
  assert_equal 3, artista.admin_obras.size
end
...

#####################################################################
# Test para comprobar que se puede crear una obra                   #
#####################################################################
test "Test debería crear una obra" do
  obra = Admin::Obra.new
```

```
    obra.titulo                = 'Lo que el viento se dejo'
    obra.referencia            = "15983"
    obra.anio_realizacion      = 2011
    obra.medidas               = '10 x 15'
    obra.precio                = 135.50
    obra.admin_coleccion_id    = 1
    obra.admin_artistum_id     = 1
    obra.admin_seccion_id      = 1
    obra.admin_tecnica_id      = 1
    obra.imagen_file_name      = 'cuadro.jpg'
    obra.imagen_content_type   = 'image/jpeg'
    obra.imagen_file_size      = 34714

    assert obra.save
  end
  ...

end
```

Código 6.1: Pruebas unitarias sobre el modelo "obras"

Podemos observar en el código que se realizan pruebas sobre: las validaciones implementadas al modelo, sobre las relaciones establecidas entre entidades *(mostradas en los ejemplos de la sección 5.3.1.2)* y por último se establece un contexto donde se comprueba la creación de una nueva instancia de la clase.

Para la ejecución de las pruebas de unidad he utilizado el comando

```
$>rake test:units
```

con el que he obtenido el siguiente resultado al aplicarla sobre la aplicación:

```
$ rake test:units

Finished in 10.86251 seconds.

145 tests, 181 assertions, 0 failures, 0 errors
```

Figura 6.1: Resultado de la ejecución de las pruebas unitarias

6.3. Pruebas de integración

El objetivo de las pruebas de integración es probar la interacción entre varios controladores. Usualmente se utilizan para probar el flujo de trabajo dentro de la aplicación.

Estas pruebas han sido creadas en el directorio *"test\integration"* de la aplicación con la siguiente estructura de nombre:

<nombre de la prueba>_test.rb

Para el desarrollo de éstas, he empleado un *lenguaje específico de dominio (DSL) en Ruby* [Dodero, Juan Manuel], utilizando módulos y bloques para definirlo.

Ello podemos observarlo en el siguiente código, el cual corresponde a un *extracto del test de integración para comprobar el correcto funcionamiento de la administración de las obras de la Galería*. Las acciones llevadas a cabo para ello vienen como comentarios.

```
class ObraTest < ActionDispatch::IntegrationTest
  # Especifico fixtures que deben cargarse
  fixtures :'users'

  # Comprueba el correcto funcionamiento de la
  # administracion de obras
  test "Gestion de obras" do
    # Crear nueva coleccion
    coleccion = Admin::Coleccion.create(
                :nombre => 'Esculturas Romanas')

    # Crear nuevo artista
    artista   = Admin::Artistum.create(
                :i_alias   => 'Marcos Torres',
                :nombre    => 'Marcos Torres Sanz',
                :curriculum => 'Nacio en Sevilla en 1955')
    ...

    # Iniciar sesion como usuario
    usuario = new_session_as(:usuario_001)

    # Anadir una nueva obra (con etiquetas)
    obra_nueva = usuario.add_obra
        :tags => 'Escultura, Estatua, Romana',
        :obra => {
                :titulo              => 'Busto de emperador',
                :referencia          => 'esc_201',
```

```
                :anio_realizacion    => 2011,
                :medidas             => '30 x 40',
                :precio              => 95.00,
                :admin_coleccion_id  => coleccion.id,
                :admin_artistum_id   => artista.id,
                :admin_seccion_id    => seccion.id,
                :admin_tecnica_id    => tecnica.id,
                :imagen_file_name    => 'busto.jpg',
                :imagen_content_type => 'image/jpeg',
                :imagen_file_size    => 478591
                }

  # Ver el listado de obras
  usuario.list_obras
  ...

  # Iniciar sesion con otro usuario
  usuario2 = new_session_as(:usuario_002)

  # Eliminar una obra
  usuario2.delete_obra obra_nueva
end


private

module ObraTestDSL
  attr_writer :name

  # Listado de obras
  def list_obras
    get "/admin/obras"
    assert_response :success
    assert_template "admin/obras/index"
  end

  # Anadir obra
  def add_obra(parameters)
    coleccion = Admin::Coleccion.find(:all).first
    artista   = Admin::Artistum.find(:all).first

    get "/admin/obras/new"
    assert_response :success
    assert_template "obras/new"
```

```
        assert_tag :tag => 'option',
                   :attributes => {:value => coleccion.id}
        assert_tag :tag => 'option',
                   :attributes => {:value => artista.id}

        post "/admin/obras", parameters

        obra = Admin::Obra.find_by_titulo(
                   parameters[:obra][:titulo])
        assert_equal parameters[:tags].split(',').size,
                   obra.tags.size

        return obra
      end

      # Borrar obra
      def delete_obra(obra)
        delete "/admin/obras/#{obra.id}"
        assert_response :redirect
        follow_redirect!
        assert_template "admin/obras/index"
      end

      ...
    end


    # Inicializar sesion
    def new_session_as(name)
      open_session do |session|
        session.extend(ObraTestDSL)
        session.name = name
        yield session if block_given?
      end
    end
  end
```

Código 6.2: Pruebas de integración para la administración de "obras"

El extracto anterior verifica que las historias de usuarios *"añadir obras"*, *"listar obras de arte"* y *"eliminar obras"* funcionan correctamente.

Se puede observar que, a diferencia de las pruebas funcionales, las pruebas de integración pueden:

- abarcar varios controladores, en el ejemplo se utilizan diferentes para crear una nueva colección, un nuevo artista y para realizar las acciones sobre las obras de arte,
- abrir varias sesiones, en el ejemplo se abren dos: una para usuario y otra para un usuario diferente: usuario2.

Para la ejecución de las pruebas de integración he utilizado el comando

```
$>rake test:integration
```

con el que he obtenido el siguiente resultado al aplicarla sobre la aplicación:

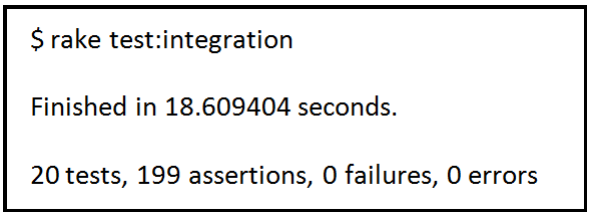


Figura 6.2: Resultado de la ejecución de las pruebas de integración

6.4. Pruebas de sistema

Mediante la realización de estas pruebas se asegura que el sistema cumple con todos los requisitos establecidos: funcionales, de almacenamiento, reglas de negocio y no funcionales.

6.4.1. Pruebas funcionales

Las clases y funciones involucradas en el controlador contienen la mayor parte de la funcionalidad que se espera de la aplicación. Según la operación a realizar, la información que se obtenga del modelo de datos será transformada y preparada para ser mostrada al usuario por pantalla.

Para garantizar que todas estas operaciones concernientes a la lógica de la aplicación se realizan correctamente, he realizado una serie de *pruebas funcionales* que podemos localizar en el directorio *"test\functional"* de la aplicación con la siguiente estructura de nombre:

```
<nombre del controlador>_controller_test.rb
```

A modo de ejemplo, muestro un extracto de las pruebas funcionales realizadas sobre el controlador *"Obras"*.

```
class Admin::ObrasControllerTest < ActionController::TestCase
  ...
  ##################################################################
  # Test para comprobar que se crea una nueva obra #
  ##################################################################
  test "Deberia poder crear una nueva obra" do
    num_obras = Admin::Obra.count
    post :create, :admin_obra => {:titulo     => 'Gato pardo',
                          :referencia         => 1,
                          :anio_realizacion   => 2010,
                          :medidas            => '20 x 20',
                          :precio             => 35.50,
                          :admin_coleccion_id => 1,
                          :admin_artistum_id  => 1,
                          :admin_seccion_id   => 1,
                          :admin_tecnica_id   => 1,
                          :imagen_file_name   => 'cuadro.jpg',
                          :imagen_content_type => 'image/jpeg',
                          :imagen_file_size   => 34714}

    assert_redirected_to admin_obra_path(assigns(:admin_obra))
    assert_equal num_obras+1, Admin::Obra.count
  end


  ##################################################################
  # Test para comprobar que se muestra una obra #
  ##################################################################
  test "Deberia poder mostrar una obra" do
    get :show, :id => @admin_obra.to_param
    assert_response :success
    assert_template 'admin/obras/show'
    assert_equal "Busto de emperador",
        assigns(:admin_obra).titulo
    assert_equal 2, assigns(:admin_obra).admin_seccion_id
  end
  ...
end
```

Código 6.3: Pruebas funcionales sobre el controlador de "obras"

Podemos observar en el extracto anterior la implementación de dos test: uno para crear una nueva obra y otro para comprobar que se muestra dicha obra.

Para la ejecución de las pruebas funcionales he utilizado el comando

```
$>rake test:functionals
```

con el que he obtenido el siguiente resultado al aplicarla sobre la aplicación:

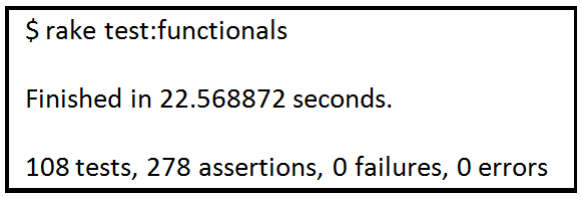


Figura 6.3: Resultado de la ejecución de las pruebas funcionales

6.4.2. Pruebas no funcionales

Estas pruebas pretenden comprobar el funcionamiento del sistema, con respecto a los requisitos no funcionales definidos en la etapa de análisis.

El tiempo de carga de una web es importante de conocer, ya que puede que la página esté cargándose de forma muy lenta haciendo que muchos usuarios se pierdan algunos contenidos, que este consumiendo muchos recursos, que tenga demasiados enlaces, o imágenes muy pesadas, por lo que es bueno hacerle un test de velocidad.

Para comprobar la *velocidad de carga* de las páginas web de la aplicación, he utilizado la herramienta online **Pingdom** (*http://tools.pingdom.com/fpt/*).

Ésta lo que hace es probar la página web por completo rastreando todo el contenido a través del código HTML (es capaz de detectar imágenes, CSS, Javascripts, RSS, Flash y frames/iframes), imitando la manera en la cual se cargaría la web normalmente. Además genera un gráfico bastante sencillo de entender donde se ve la evolución de carga en segundos de cada uno de los archivos de la web, el cual nos permite saber cuál es el origen de una posible carga lenta y a qué archivo se debe. Posee un funcionamiento bastante sencillo, ya que tan solo hay que ingresar la URL para obtener el test.

En la *tabla 6.1* muestro los tiempos de carga de diferentes páginas web que componen la aplicación, pudiéndose observar que tardan menos de un segundo, aunque hay algunas que se podrían optimizar más.

A modo de ejemplo voy a enseñar un análisis completo para la url `http://gadeirart.heroku.com/catalogo` *(figura 6.5)*.

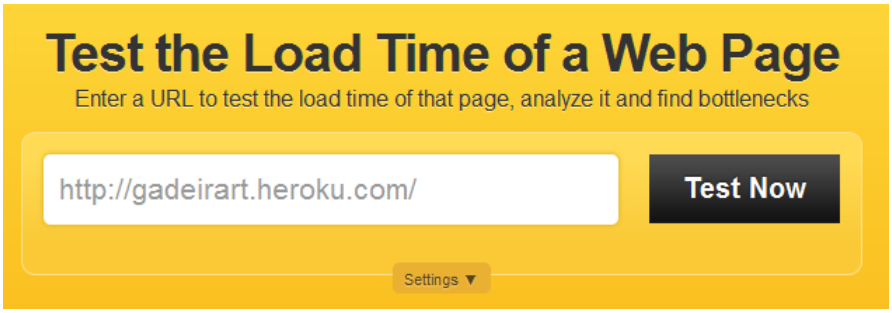


Figura 6.4: Herramienta "Pingdom"

Web	URL	Perf. grade	Request	Load time (ms)	Page size (kB)
Inicio	/inicio	58/100	31	775	456.7
Quienes somos	/galeria/nosotros/1	56/100	26	600	386.6
Catálogo	/catalogo	57/100	32	674	420.5
Condiciones	/galeria/condiciones/index_usuario	55/100	24	340	249.7
Contactar	/contactar	56/100	25	661	253.5
Obra	/catalogo/show/11	56/100	26	854	345.0
Imagen Obra	/catalogo/show_photo/11	56/100	26	729	522.7
Artista	/galeria/artista/3	54/100	26	632	251.9
Obras recientes	/catalogo/latest	59/100	37	823	390.8
Búsqueda	/catalogo/search	58/100	32	711	436.5
Búsqueda concreta	/catalogo/search?utf8...	60/100	29	995	405.5
Registrarse	/users/new	55/100	24	345	249.6
Iniciar Sesión	/user_sessions/new	55/100	24	415	249.8
Olvidó contraseña?	/password_resets/new?locale=es	56/100	25	664	251.1
Cesta de la compra	/carrito/index	53/100	25	365	249.8
Facturar	/checkout/index	100/100	1	603	957.0
Administración	/admin/administracions	55/100	24	286	249.2
Listado Artistas	/galeria/artista	49/100	30	717	256.8
Mostrar artista	/galeria/artista/3	54/100	26	290	251.9
Editar artista	/galeria/artista/3/edit	56/100	27	714	254.3
Traducir artista	/admin/traduccionartista/3/edit	57/100	26	348	252.6
Listado colecciones	/admin/coleccions	50/100	29	675	255.7
Listado sesiones	/admin/seccions	50/100	29	735	255.4
Listado técnicas	/admin/tecnicas	50/100	29	640	255.6
Listado obras	/admin/obras	51/100	35	873	333.9
Pedido	/admin/pedido/show?id=1	55/100	25	487	251.7

Tabla 6.1: Velocidad de carga dentro de "http://gadeirart.heroku.com/"

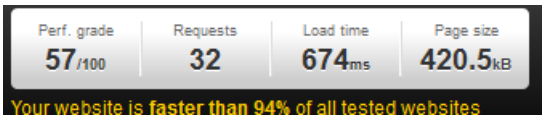


Figura 6.5: Velocidad de carga para la url "http://gadeirart.heroku.com/catalogo"

Podemos ver que esta herramienta nos enseña visualmente, en un gráfico de barras (figura 6.6), el tiempo, mostrándonos la lista de objetos en el orden en que se van cargando jerárquicamente. Esto nos permite ver los objetos que están siendo linkeados. Un ejemplo de los gráficos obtenidos pueden contemplarlo a continuación:

Cada test de la*figura 6.7* muestra estadísticas generales acerca de la carga de la página, así como el número total de objetos, tiempo de carga y tamaño de todos los objetos incluidos.

He realizado un segundo análisis y para estas pruebas he optado por *New Relic* (`https://newrelic.com/`), la cual es una plataforma para el monitoreo y análisis del rendimiento de aplicaciones que es ofrecida como un servicio en la web.

Para poder hacer uso de ella he añadido:

- una nueva gema, "**newrelic_rpm**", en el fichero *"Gemfile"* de la aplicación y
- el archivo de su configuración "**newrelic.yml**" en el directorio *"config"*, obtenido a través de la web de la plataforma.

A partir de este momento, los agentes del lenguaje envían métricas de la aplicación monitoreada a *New Relic* cada minuto, mientras se navega por las diferentes páginas de la web. Para ver esta información de rendimiento, incluyendo un análisis detallado de las sentencias SQL, hay que abrir *New Relic* en el navegador (`http://localhost:3000/newrelic`) y obtenemos la información de la *figura 6.8*.

También se pueden visualizar los detalles, por ejemplo, para la cuarta de las URLs que aparecen, `/es/galeria/artista`, en la *figura 6.8*. Ello lo vemos representado en la *figura 6.9*.

Y contemplar, por ejemplo para la misma URL, el rendimiento obtenido por las sentencias SQL que ejecuta *(figura 6.10)*.

Un resumen lo obtenemos en la siguiente *imagen 6.11*.

Figura 6.6: Velocidad de carga para la url “http://gadeirart.heroku.com/catalogo”

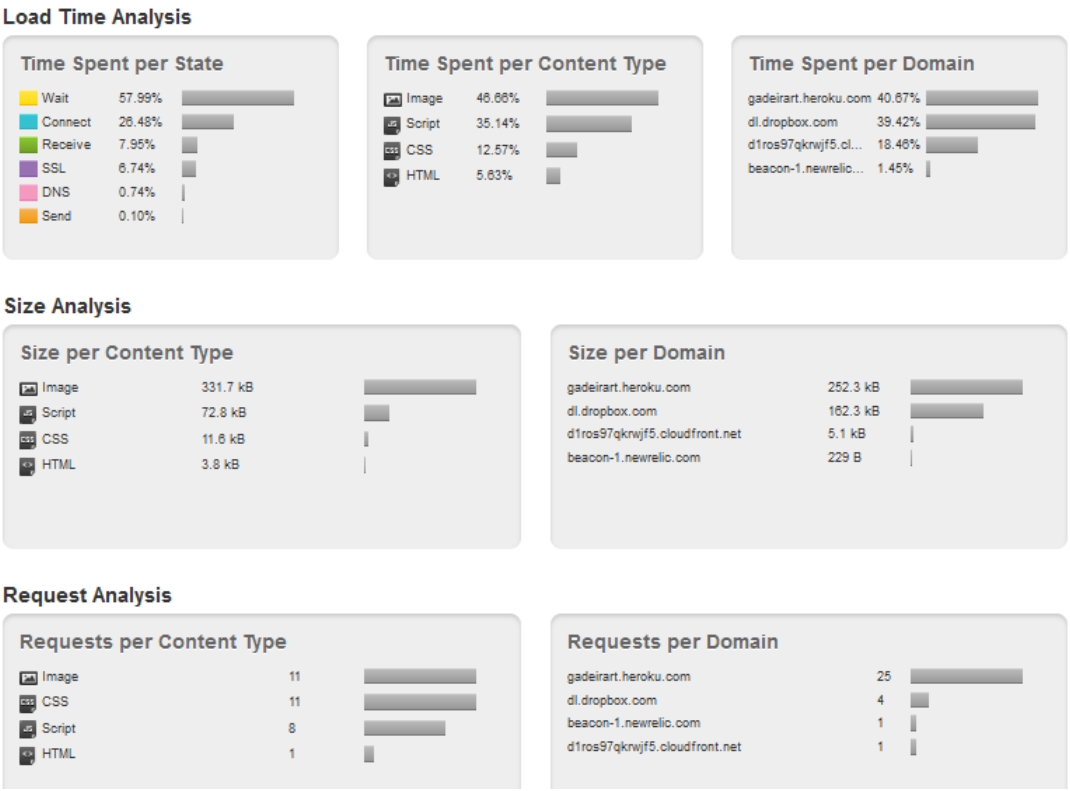
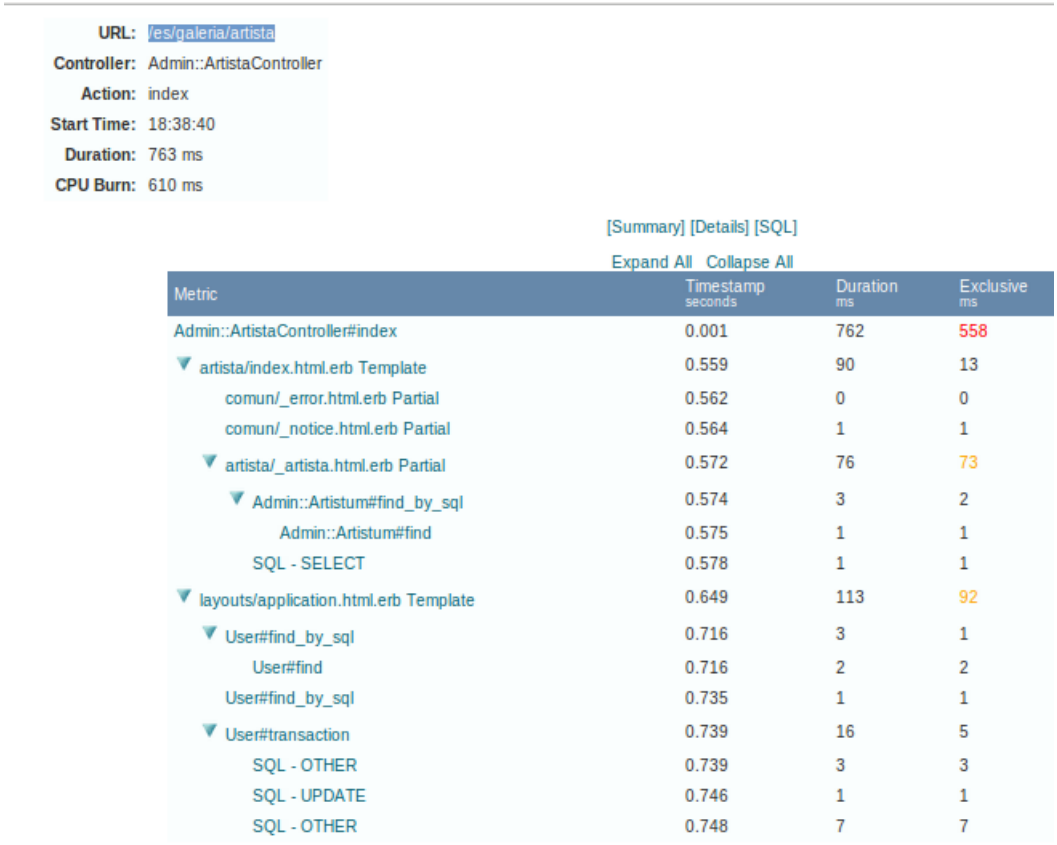


Figura 6.7: Análisis de la url "http://gadeirart.heroku.com/catalogo"

Figura 6.8: Monitorización de la aplicación en modo de desarrollo



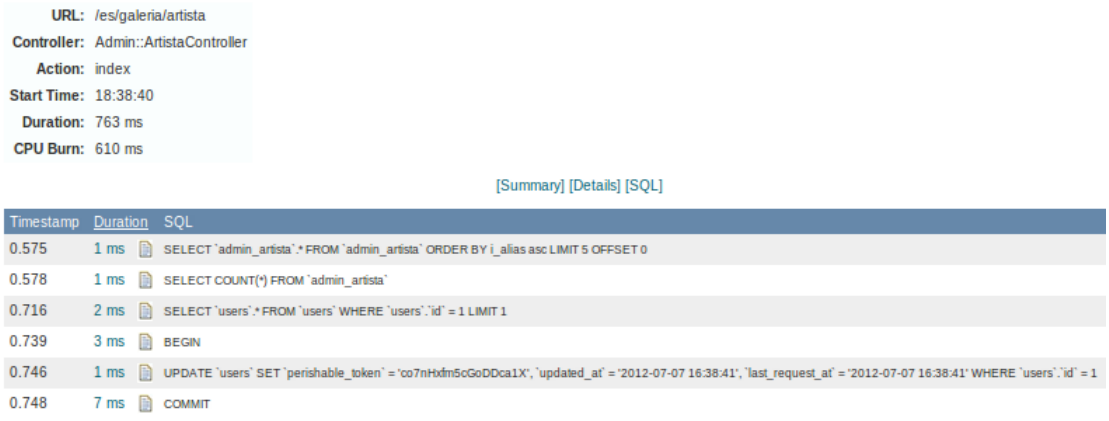
URL: /es/galeria/artista
Controller: Admin::ArtistaController
Action: index
Start Time: 18:38:40
Duration: 763 ms
CPU Burn: 610 ms

[Summary] [Details] [SQL]

Expand All Collapse All

Metric	Timestamp seconds	Duration ms	Exclusive ms
Admin::ArtistaController#index	0.001	762	558
artista/index.html.erb Template	0.559	90	13
comun/_error.html.erb Partial	0.562	0	0
comun/_notice.html.erb Partial	0.564	1	1
artista/_artista.html.erb Partial	0.572	76	73
Admin::Artistum#find_by_sql	0.574	3	2
Admin::Artistum#find	0.575	1	1
SQL - SELECT	0.578	1	1
layouts/application.html.erb Template	0.649	113	92
User#find_by_sql	0.716	3	1
User#find	0.716	2	2
User#find_by_sql	0.735	1	1
User#transaction	0.739	16	5
SQL - OTHER	0.739	3	3
SQL - UPDATE	0.746	1	1
SQL - OTHER	0.748	7	7

Figura 6.9: Detalles de la URL "/es/galeria/artista"



URL: /es/galeria/artista
Controller: Admin::ArtistaController
Action: index
Start Time: 18:38:40
Duration: 763 ms
CPU Burn: 610 ms

[Summary] [Details] [SQL]

Timestamp	Duration	SQL
0.575	1 ms	SELECT `admin_artista`.* FROM `admin_artista` ORDER BY i_alias asc LIMIT 5 OFFSET 0
0.578	1 ms	SELECT COUNT(*) FROM `admin_artista`
0.716	2 ms	SELECT `users`.* FROM `users` WHERE `users`.`id` = 1 LIMIT 1
0.739	3 ms	BEGIN
0.746	1 ms	UPDATE `users` SET `perishable_token` = 'co7nHxfm5cGoDDca1X', `updated_at` = '2012-07-07 16:38:41', `last_request_at` = '2012-07-07 16:38:41' WHERE `users`.`id` = 1
0.748	7 ms	COMMIT

Figura 6.10: Detalles de las sentencias SQL en la URL "/es/galeria/artista"

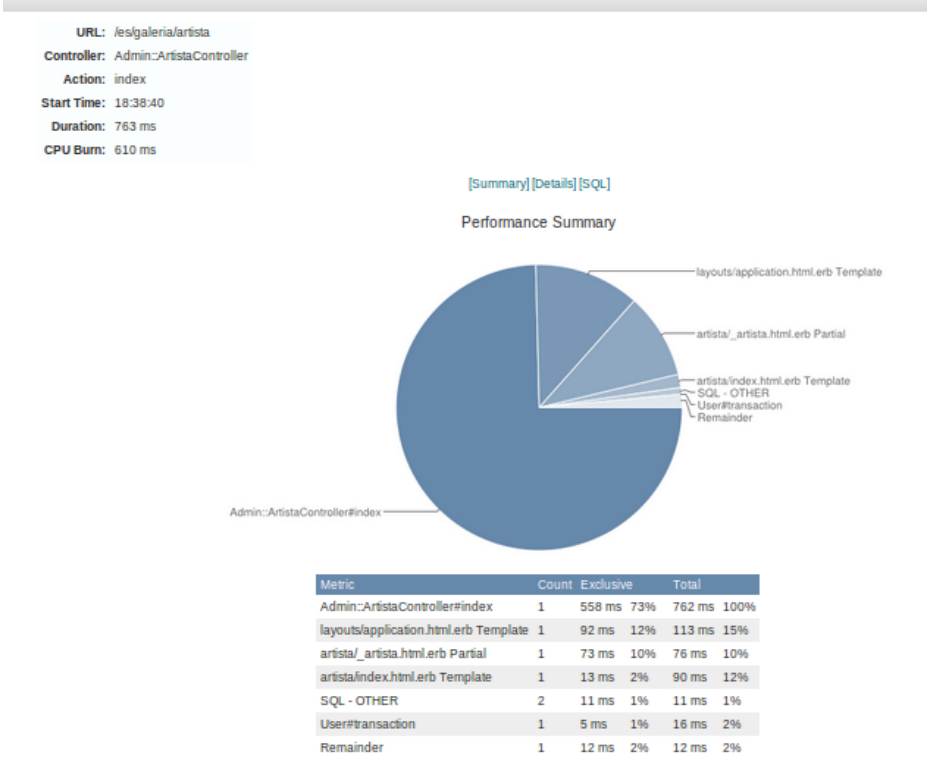


Figura 6.11: Resumen para la URL "`/es/galeria/artista`"

Podemos observar en las imágenes anteriores, que hay rendimientos, marcados en rojo, los cuales deberían ser optimizados para obtener unos mejores resultados en la aplicación, esto lo voy a dejar como una de las mejoras para realizar de la aplicación. También comentar que dependiendo del navegador y del hardware que se utilice se pueden obtener diferentes rendimientos.

6.5. Pruebas de aceptación

El objetivo de estas pruebas es validar el sistema de desde el punto de vista funcional y operativo.

Para la realización de estas pruebas, he solicitado al cliente[1] y a varios usuarios ajenos al desarrollo, que accedan a la web desde sus entornos de trabajo, realicen varias pruebas sobre la aplicación y verifiquen su eficacia.

El resultado de estas pruebas ha sido un producto exitoso, logrando la aprobación por parte de todos.

De esta manera, el producto está listo para su paso a producción.

[1] *Cliente:* Galerista o Artista

Parte III

Epílogo

En esta última parte de la memoria se recogen las conclusiones y los manuales necesarios para el manejo de la tienda virtual.

Capítulo 7

Manual de usuario

7.1. Introducción

Este manual proporciona una guía para el *usuario* de la tienda virtual de la *Galería de Arte "GadeirArt"*, detallándose las funcionalidades de la aplicación. Los lectores no requieren ningún conocimiento acerca de programación ni desarrollo de software.

Veremos el funcionamiento de la aplicación desde los siguientes roles:

- **Administrador** cuyo objetivo será administrar la aplicación web.
- **Usuario final** cuyo fin será visualizar las obras de arte de la Galería y comprarlas en caso de estar interesado.

7.2. Características

La aplicación consiste en un sistema web formado por:

1. Un *Front Office*, con el que interactuarán los usuarios, formado por una Tienda Online. Aquí podrán acceder al catálogo de las Obras de Arte y realizar sus compras.
2. Un *Back Office*, espacio restringido que permitirá al administrador gestionar toda la información del sistema.

Las funcionalidades que ofrece el sistema están definidas en la sección 3.2.1 *"Historias de usuario"* y son las siguientes:

- *Gestión de Artistas*: La *Galería* poseerá información de cada uno de los *Artistas*.
- *Gestión de Colecciones*: Poseerá información de cada una de las *Colecciones de Arte* que posea.
- *Gestión de Obras*: Poseerá información de cada una de las *Obras de Arte* que tenga en su inventario.
- *Gestión de Secciones*: La Galería de Arte clasificará sus obras en diferentes *Secciones*.
- *Gestión de Técnicas*: Cada una de las *Secciones* posee una serie de *Técnicas* con las que se realizan las *Obras de Arte*.
- *Gestión de Catálogo*: El *Catálogo* servirá para que el usuario[1] pueda consultar la relación de obras que posee la *Galería*, así como la ficha técnica de cada una de ella.
- *Gestión de Cesta de la Compra*: En ella el cliente podrá ir añadiendo cada una de las obras de arte que desee comprar.
- *Gestión de Etiquetado*: El *Etiquetado* tendrá dos fines para la *Galería de Arte*:
  - Poder tener las obras de arte clasificadas por categoría de forma que el usuario pueda acceder a dicha clasificación.
  - Recomendar obras relacionadas cuando los usuarios estén navegando por la ficha técnica de una obra.
- *Gestión de Seguridad*: La *Galería* necesita que solo el usuario *Administrador* sea el que pueda administrar la aplicación evitando así algún que otro mal para la empresa. Para realizar la compra los visitantes deberán identificarse como clientes, o registrarse si todavía no tienen cuenta.
- *Gestión de Facturación*: Posee un sistema de procesamiento para los pedidos de sus clientes a través de una pasarela de pago.
- *Gestión de Internacionalización*: Las páginas pueden verse tanto en español como en inglés, para así poder acaparar una mayor clientela.
- *Gestión de Quienes Somos*: Mediante este página el usuario accederá a la descripción de la actividad de la galería.
- *Gestión de Condiciones*: Mediante este menú el usuario accederá a las condiciones generales de la galería.

[1] *Usuario*: Visitante o Cliente

7.3. Requisitos previos

Los usuarios deberán disponer de un ordenador que posea una tarjeta de red o una tarjeta de red inalámbrica y un punto de acceso que les permita una conexión a Internet para poder acceder a la aplicación.

También necesitarán la utilización de un navegador web para que la aplicación pueda ser visualizada, no siendo necesario el uso de uno específico debido a que la aplicación esta adaptada para poder ser visualizada en la mayoría de los navegadores disponibles.

7.4. Utilización

Cualquier navegante podrá acceder a la web desde un navegador a través de la dirección: http://gadeirart.heroku.com/.

No hacer falta estar registrado para visualizar los contenidos de la tienda, pero sí será necesario para poder comprar.

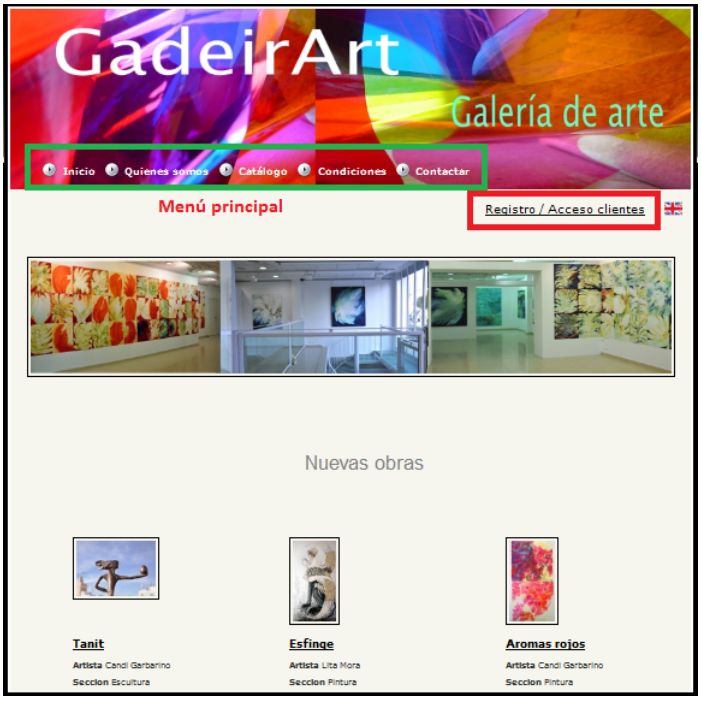


Figura 7.1: Captura del página de inicio

En cada una de las páginas de la web existirá un menú principal desde el que se puede acceder a todas las funcionalidades presentes en la aplicación según el rol que tenga asignado el usuario.

7.4.1. Administrador

El usuario *administrador* es el encargado de gestionar toda la información del sistema. Sus funcionalidades se ejecutan en la interfaz de administración.

7.4.1.1. Logarse como administrador y modificar sus datos

Para tener acceso a ella, el *administrador* deberá logarse desde el enlace *"Registro/Acceso clientes"* (figura 7.1) con los siguientes datos iniciales:

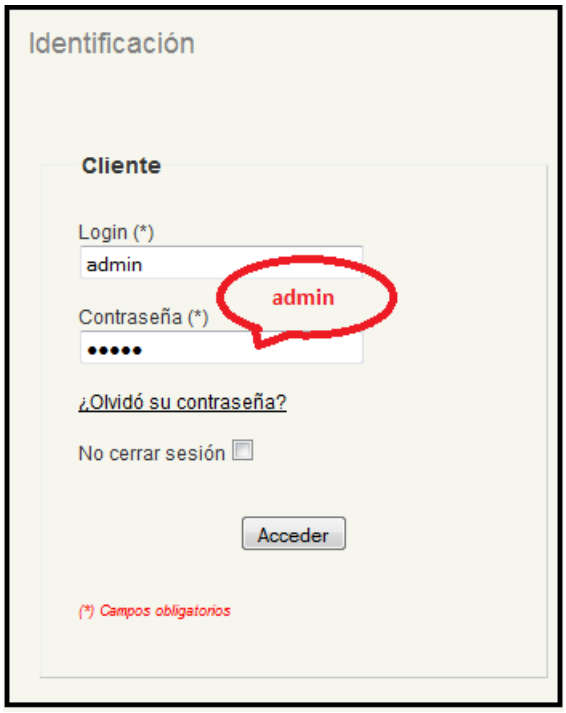


Figura 7.2: Captura de la identificación del administrador

Una vez registrado le aparecerá una barra de menú (figura 7.3), similar al del resto de usuarios, con una opción más: **"Administración"**, desde la que podrá gestionar toda la información de la web.

También podemos ver en la imagen 7.3 que aparece un link llamado "Mi cuenta" que mostrará al *administrador* sus datos (figura 7.4). Desde él podrá *modificar su clave de acceso* haciendo click sobre el botón "Editar" y rellenando el formulario que se presenta *(figura 7.5)*.

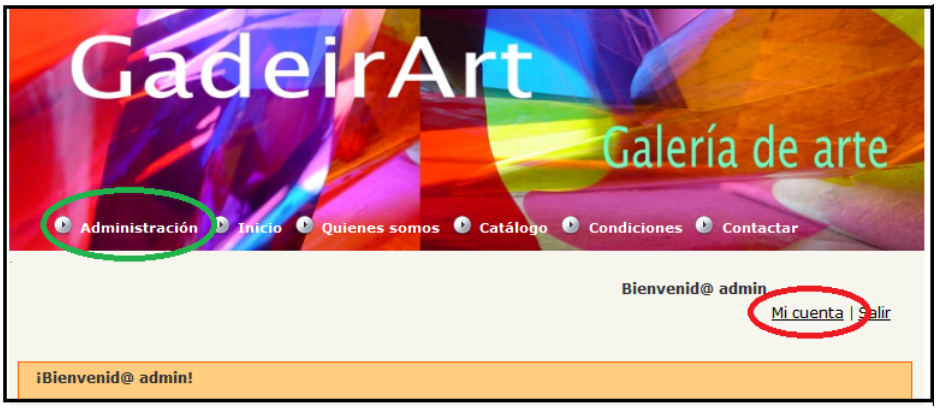


Figura 7.3: Captura del menú del administrador

Figura 7.4: Captura de los datos del administrador

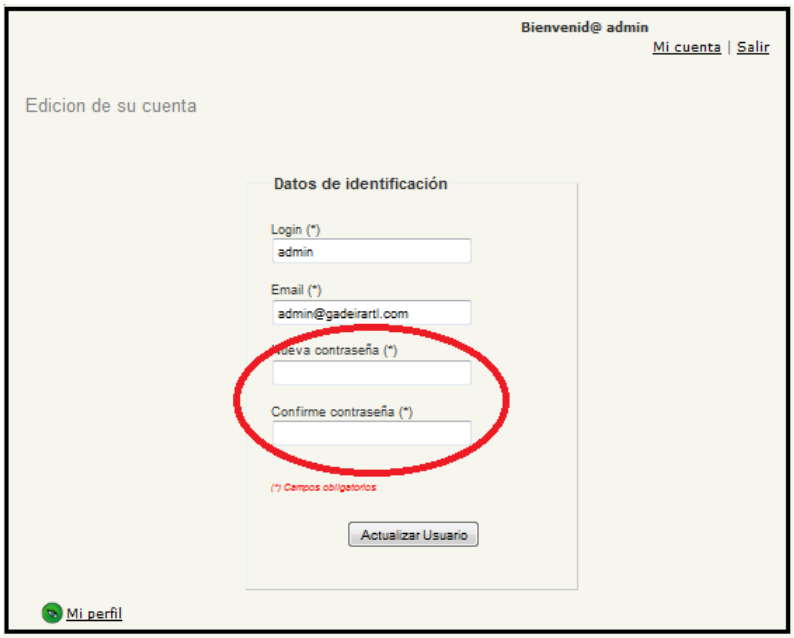


Figura 7.5: Captura de la edición de los datos del administrador

7.4.1.2. Panel de administración

Desde el "panel de administración" (figura 7.6), el *administrador* tendrá acceso a una serie de funcionalidades:

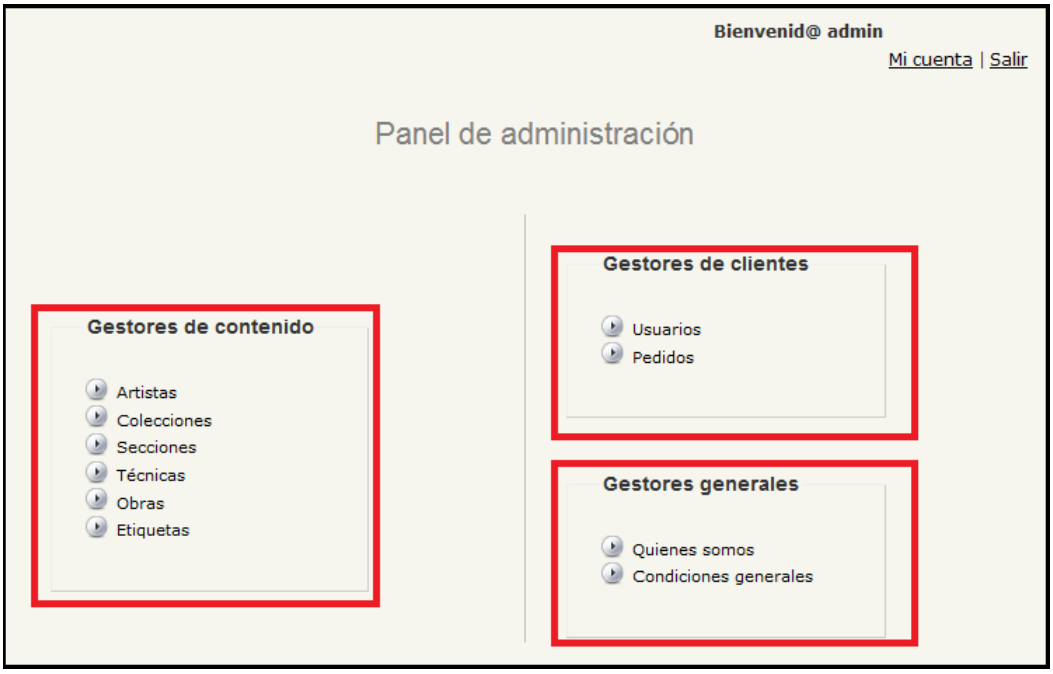


Figura 7.6: Captura del panel de administración

El *panel de administración* aparece dividido en tres partes:

- **Gestores de contenido**
  - **Artistas**: Contendrá cada una de las funcionalidades de los artistas de la Galería.
  - **Colecciones**: Desde aquí se podrá administrar cada una de las colecciones de las obras de arte.
  - **Secciones**: Las diferentes categorías de las obras de arte (dibujo, escultura, pintura, fotografía...)
  - **Técnicas**: Cada una de las técnicas en que se dividen las secciones anteriores.
  - **Obras**: Obras de arte que posee la Galería.
  - **Etiquetas**: Etiquetas con las que están clasificadas las obras de arte.
- **Gestores de clientes**
  - **Usuarios**: Usuarios registrados en la Galería y su información.
  - **Pedidos**: Cada uno de los pedidos realizados a la Galería por los clientes junto con su información.
- **Gestores generales**
  - **Quienes somos**: La funcionalidad de este apartado es la de describir la actividad de la Galería.
  - **Condiciones generales**: Sirve para redactar cada una de las condiciones generales de la Galería.

Dentro de ellos, el usuario *administrador* podrá administrar todas las funcionalidades de cada uno de los apartados de la galería haciendo click sobre la opción que desee en el *panel de administración*.

7.4.1.3. Listados

Cuando hacemos click en algunas de las opciones de los gestores anteriormente comentados, accedemos al listado de la opción elegida. Por ejemplo, el *listado de los artistas* se obtiene haciendo click sobre el botón *"Artistas" (figura 7.7)*.

Figura 7.7: Captura de la pantalla de administración de artistas

Voy a explicar la funcionalidad de cada uno de los botones que vemos en la pantalla anterior u otras.

- **Botón "Nuevo"**: Sirve para dar de alta a nuevos artistas de la Galería de Arte. Para ello tendrá que rellenar cada uno de los campos solicitados en el formulario que se presenta.
- **Botón "Mostrar"**: Podrá visualizar los datos del artista.
- **Botón "Editar"**: Sirve para editar y modificar la información del artista.
- **Botón "Eliminar"**: Para eliminar el artista seleccionado.
- **Botón "Traducir al inglés"**: Desde el que podrá traducir al inglés, a través de un formulario, cierta información del artista.
- **Botón "Retroceder"**: Retrocede a la página anterior.

Dar un nuevo alta

El *administrador* podrá dar un nuevo alta (de artista, de obras, colecciones...) haciendo click en el listado, o dentro de cualquier otra pantalla, sobre el botón "Nuevo". Aparecerá una nueva ventana que contiene un formulario en el que hay que rellenar todos los campos que sean obligatorios y tras pulsar el botón se realizará un nuevo alta *(figura 7.8)*.

Figura 7.8: Captura de la pantalla de dar de alta una nueva colección

Mostrar

Para tener acceso a la información sobre las nuevas altas haremos click sobre el botón *"Mostrar"* que aparece en el listado o dentro de cualquier otra pantalla (figura 7.9).

En esta pantalla también tenemos una serie de botones que podemos utilizar para la administración.

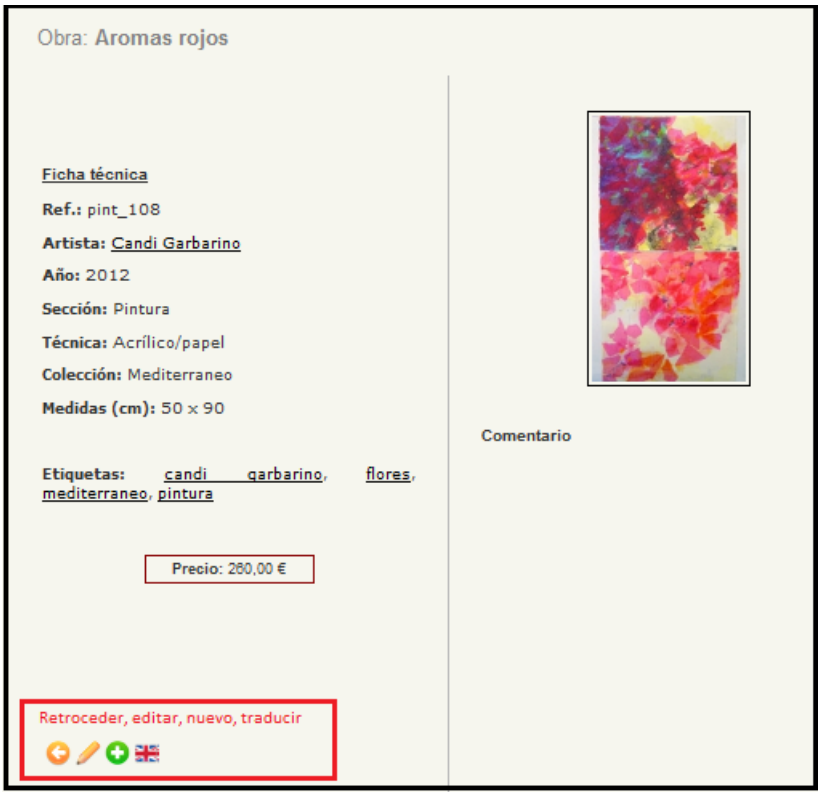


Figura 7.9: Captura de la pantalla que muestra los datos de una obra

Editar

Para editar la información tan solo hay que pulsar sobre el botón *"Editar"* que aparece en el listado o dentro de cualquier otra pantalla. Aparecerá una nueva pantalla conteniendo la información, pudiendo modificar la que deseemos. Para que las modificaciones se lleven a cabo habrá que hacer click sobre el botón *"Actualizar"*.

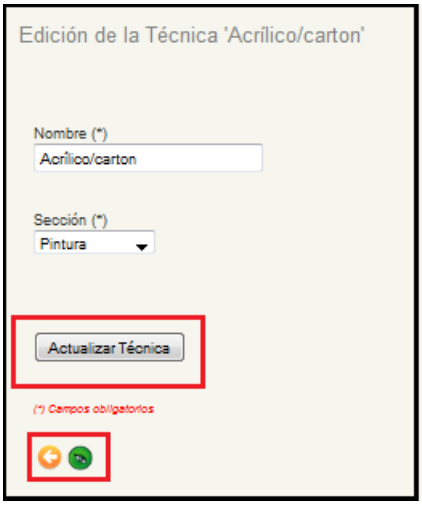


Figura 7.10: Captura de la pantalla de edición de una técnica

Eliminar

Para eliminar debemos hacer click sobre el botón *"Eliminar"* que aparece en el listado y aparecerá una ventana emergente que nos preguntará si deseamos o no hacer la eliminación.

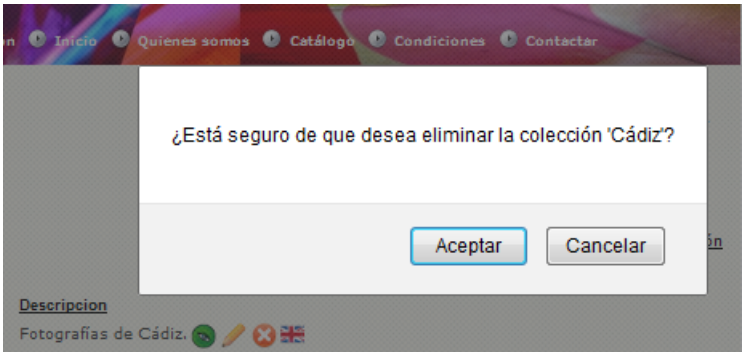


Figura 7.11: Captura de la pantalla de confirmación de eliminación

Traducir al inglés

El *administrador* podrá traducir algunos de los campos para que la información pueda ser transmitida al usuario final en dos idiomas: español e inglés.

Para ello tendrá que pulsar sobre el botón *"Traducir al inglés"* que aparece en el listado *(figura 7.9)* o dentro de cualquier otra pantalla. Una vez traducido y tras pulsar el botón *"Traducir" (figura 7.12)*, podrá ver como le ha quedado la traducción.

Figura 7.12: Captura de la pantalla de traducción

7.4.1.4. Búsqueda de obras de arte

En el *listado de las obras de arte* aparecerá la siguiente pantalla similar a la de la *figura 7.7*, pero se observa que tiene una parte nueva, la *búsqueda*:

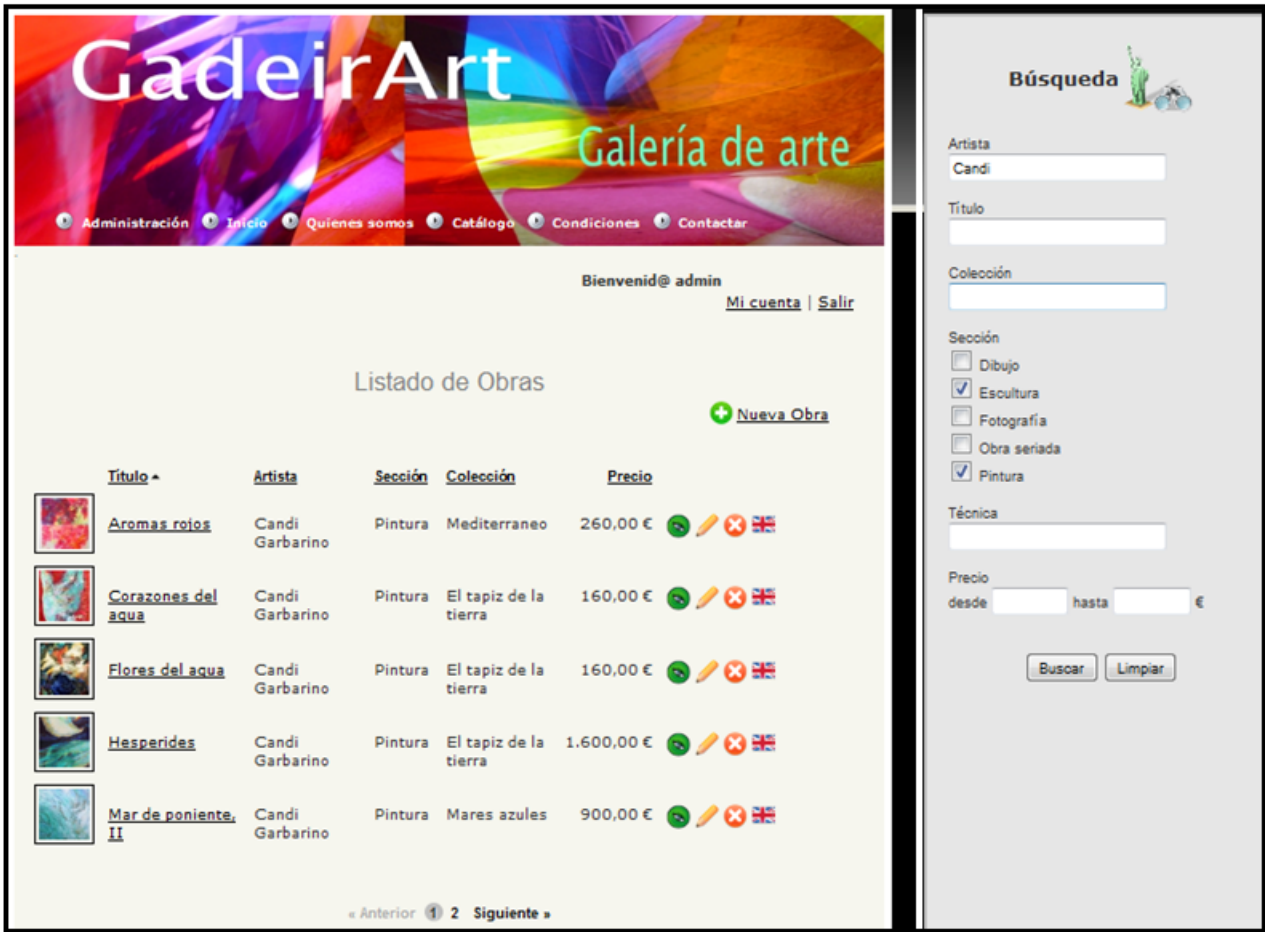


Figura 7.13: Captura de ejemplo de búsquedas de obras

donde el *administrador* podrá realizar búsquedas rellenando cualquiera de los campos del formulario que aparecen a la derecha de la página *(figura 7.13)* o seleccionando cualquiera de las secciones. Para que se lleve a cabo, se deberá pulsar sobre el botón *"Buscar"* y a continuación aparecerán todas las obras de arte que cumplen con los criterios seleccionados.

El botón *"Limpiar"* sirve para eliminar las opciones elegidas para la búsqueda.

7.4.1.5. Gestores de clientes: Pedidos

Para acceder a la administración de los *pedidos* de los clientes, el administrador deberá hacer click sobre la opción *"Pedidos"* del panel de administración (figura 7.6).

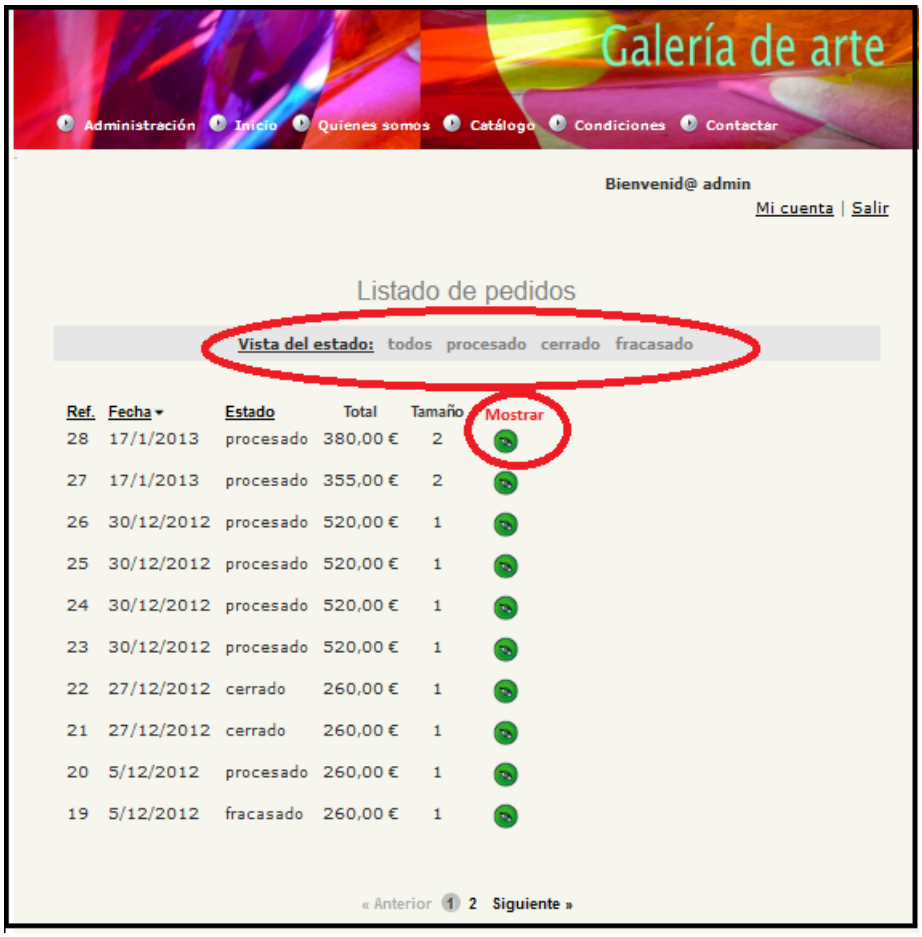


Figura 7.14: Captura del listado de pedidos de los clientes

En la figura 7.14, vemos que presenta varias *"Vistas del estado": todos, procesado, cerrado, fracasado*. Según la vista que seleccione aparecerá el listado de los pedidos de esa opción.

- **Todos**: Mostrará todos los pedidos de los clientes.
- **Procesado**: Pedidos que están en proceso y aún no han sido enviados al cliente.
- **Cerrado**: Pedidos que ya se han enviado al cliente.
- **Fracasado**: Pedidos que por alguna razón han fracasado en su registro.

El usuario *Administrador* podrá visualizar toda la información referente al pedido de un cliente. Para ello tan solo tendrá que hacer click sobre el botón *"Mostrar"* de la pantalla 7.14.

Figura 7.15: Captura pantalla de información de un pedido

El botón *"Cerrar pedido"* se pulsará una vez se haya enviado el pedido al cliente, con lo que el estado del pedido cambiará a *"Cerrado"* como podemos ver en la *figura 7.14*.

7.4.2. Usuario de la tienda

7.4.2.1. Registrarse en la tienda virtual

Un usuario anónimo tiene la posibilidad de registrarse como "cliente" de la tienda virtual, para ello deberá acceder a la pantalla de registro desde el enlace *"Registro/Acceso clientes"* que aparece en cualquier pantalla.

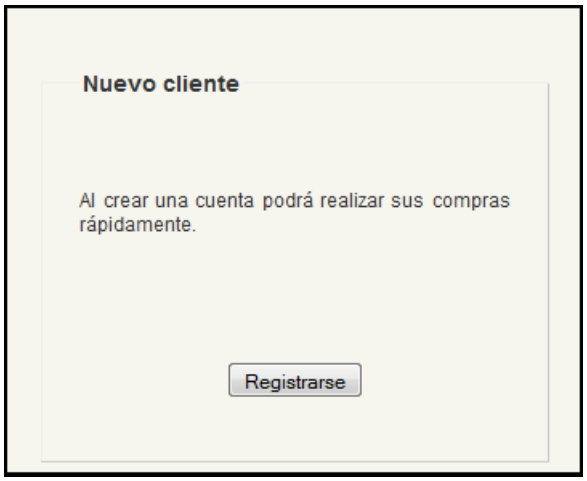


Figura 7.16: Captura pantalla de registro de nuevo cliente

En la *figura 7.16* deberá hacer click en el botón *"Registrarse"*, teniendo acceso a la siguiente pantalla:

Figura 7.17: Captura pantalla de datos de identificación para el registro

donde es obligatorio insertar todos los campos marcados con "*" y tras completar la información solicitada se dará de alta al usuario en la tienda virtual. Éste recibirá un correo electrónico de *notificación de registro*.

Figura 7.18: Captura pantalla conteniendo la información del usuario registrado

7.4.2.2. Modificación de los datos de usuario

Cualquier usuario que se encuentre logado en la web puede modificar sus datos. Para ello deberá pulsar el link *"Mi cuenta"* de cualquier pantalla y luego, en la pantalla que aparece, sobre el botón *"Editar"* (figura 7.18).

Al pulsar accedemos a la zona donde el usuario puede modificar todos sus datos. Para ello basta con modificar los datos disponibles y pulsar sobre el botón *"Actualizar usuario"* (figura 7.19).

7.4.2.3. Catálogo

Para acceder al catálogo de la tienda virtual de la Galería, deberá hacer click (figura 7.20) sobre el menú *"Catálogo"* que aparece en cualquiera de las páginas.

7.4.2.4. Obras recientes de la galería

Podemos tener acceso a las últimas adquisiciones de la Galería haciendo click sobre el link *"Obras recientes"* que aparece dentro del catálogo *(figura 7.21)*.

Figura 7.19: Captura pantalla de edición de datos de identificación del usuario

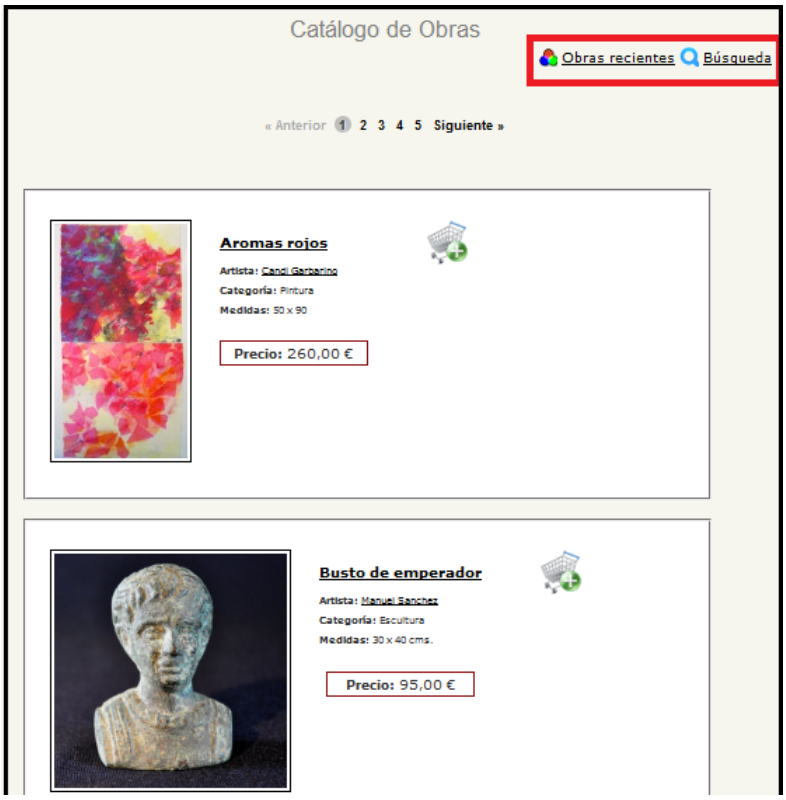


Figura 7.20: Captura del catálogo

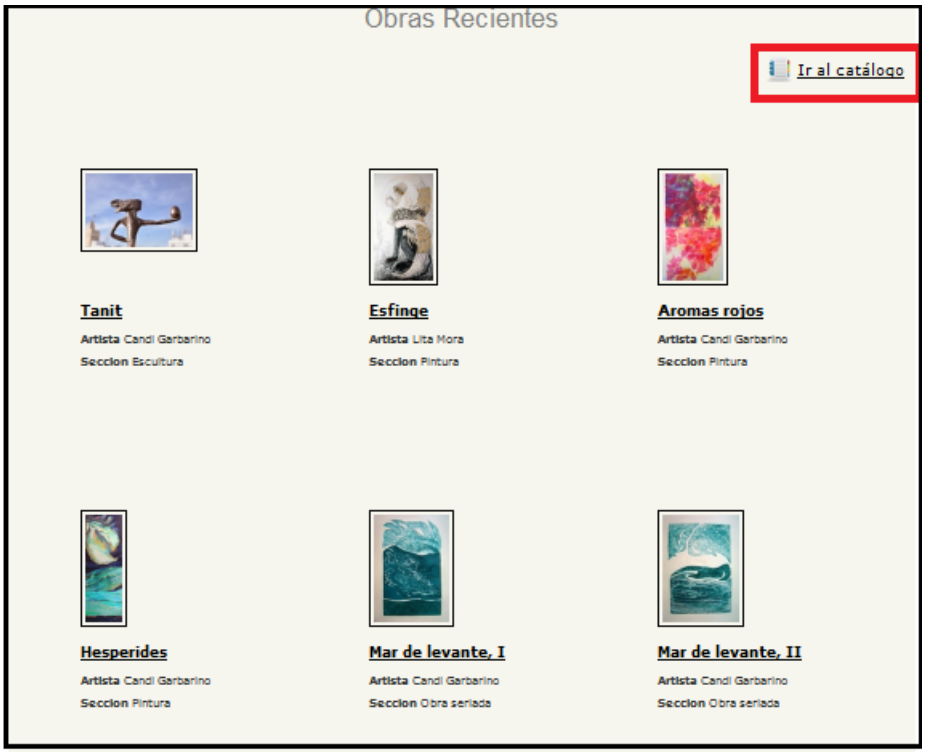


Figura 7.21: Captura de las obras recientes de la galería

7.4.2.5. Búsqueda de obras de arte

La *búsqueda de obras de arte* se realizará desde el link *Búsqueda* que nos encontramos dentro del *catálogo de obras (figura 7.20)*.

En la pantalla de figura 7.22 podremos realizar las búsquedas rellenando cualquiera de los campos del formulario que aparecen a la derecha de la página o seleccionando cualquiera de las secciones. Para que se lleve a cabo, se deberá pulsar sobre el botón *"Buscar"* y a continuación aparecerán todas las obras de arte que cumplen con los criterios seleccionados.

El botón *"Limpiar"* sirve para eliminar las opciones elegidas para la búsqueda.

7.4.2.6. Consulta y selección de obras de arte

En este apartado se definen las posibles acciones que se pueden realizar sobre las obras de arte de la Galería obtenidos a través de la búsqueda o en el catálogo de la Galería.

Figura 7.22: Captura pantalla de búsqueda de obras de arte

Obras que posee la galería

Independientemente de cómo se han obtenido los listados de obras de arte que posee la galería, siempre se muestran de la siguiente forma:

Figura 7.23: Captura la apariencia de una obra de arte

y con los siguientes datos que cabe destacar:

- *Foto de la obra*: La foto puede ampliarse pulsando sobre ella.
- *Título de la obra*: Pulsando sobre él tenemos acceso a toda la información de la obra de arte.

Figura 7.24: Captura de la pantalla con la información de la obra

- *Artista*: Haciendo click sobre el artista podrá visualizar información sobre éste.
- *Cesta de la compra*: Para comprar la obra hay que añadirla a su cesta de la compra y para ello bastara con hacer click sobre ella.

Ver información de la obra

Para ver la información de la obra basta con pulsar sobre el *título* de la obra que aparece en la pantalla como la de la figura 7.23.

Pulsando sobre alguna de las etiquetas de la obra accedemos a un listado de todas las obras que contienen dicha etiqueta.

Podemos observar que aparte de ver la información sobre la obra nos aparece a la derecha una serie de recomendaciones de obras *(figura 7.24)*.

Ver foto de la obra ampliada

Para poder ver las fotos ampliada basta con pulsar sobre la imagen, se abrirá una nueva ventana donde se visualizará la imagen de la obra de arte en un tamaño superior *(figura 7.25)*.

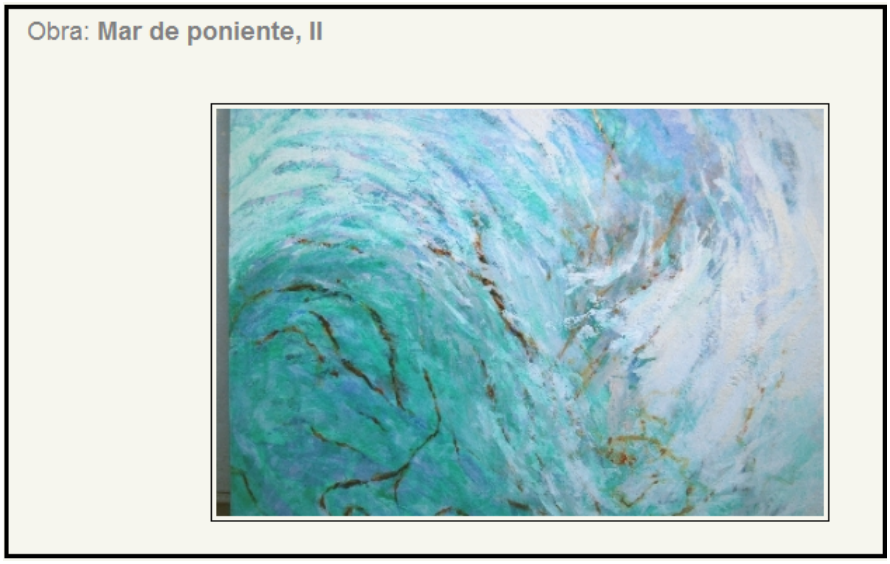


Figura 7.25: Captura de la imagen de la obra ampliada

Ver información del artista

Para ello basta con hacer click sobre el *nombre del artista* que aparece en la pantalla como la de la *figura 7.23* y se mostrará una nueva ventana con toda la información sobre éste.

Añadir una obra a la cesta de la compra

Para añadir una obra de arte a nuestra cesta de la compra tenemos que pulsar sobre la imagen del carrito como el que vemos en la figura 7.23.

En la *figura 7.26* se nos muestra *la cesta de la compra* conteniendo las obras que deseamos comprar junto con su importe.

7.4.2.7. Cesta/carrito de la compra

Desde el link *"Mi cesta"*, disponible en cualquier pantalla si se está logeado, se puede acceder a ver la *cesta de la compra* (figura 7.27).

El usuario puede seguir añadiendo líneas pulsando sobre el botón *"Ir al catálogo"* o eliminarlas pulsando sobre el botón *"Borrar"*.

Figura 7.26: Captura de la cesta de la compra

7.4.2.8. Realizar el pedido

Para realizar el pedido hay que pagar el contenido de la cesta, para acceder a esta zona hay que pulsar sobre el botón *"Comprar"* que se encuentra en la *cesta de la compra*.

Logarse en la web

Para poder realizar el pedido hay que estar logado en la web.

Para ello tenemos el enlace *"Registro/Acceso clientes"* que aparece en cualquier pantalla y que nos envía a la página donde poder realizarlo. A esta página (figura 7.28) también se accede si el usuario no esta logado y pulsar sobre el botón *"Comprar"* en la *cesta de la compra*.

Introducción de datos personales y confirmación del pedido

Si el usuario está logado en la tienda virtual de la Galería, sus datos aparecen automáticamente y estos podrán ser modificados en caso de que los datos de facturación varíen.

Figura 7.27: Captura de la cesta de la compra

Figura 7.28: Captura la página para logarse

Tras indicar todos los datos necesarios hay que seleccionar la *forma de pago* con la que se desea pagar y pulsar el botón *"Continuar"* para realizar el pedido.

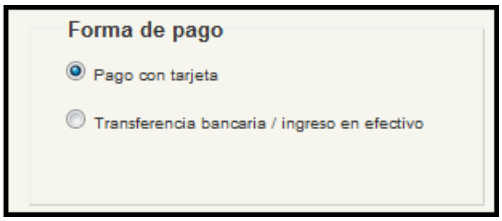


Figura 7.29: Captura de la forma de pago

Tras rellenar los datos, si son necesarios, para la forma de pago y pulsar sobre el botón *"Realizar el pedido"*, se muestra un resumen de los datos indicados para que el usuario confirme el pedido pulsando sobre el botón *"Comprar"*. De esta manera el pedido ya se ha enviado a la Galería, la cual recibirá un email con los datos del pedido y el usuario recibirá otro de confirmación de pedido.

Forma de pago

Como se muestra en la figura 7.29, el usuario tiene varias formas de pago para el pedido.

- **Pago con tarjeta**: Si selecciona esta opción aparecerá la siguiente pantalla en la que deberá rellenar los datos de su tarjeta de crédito.

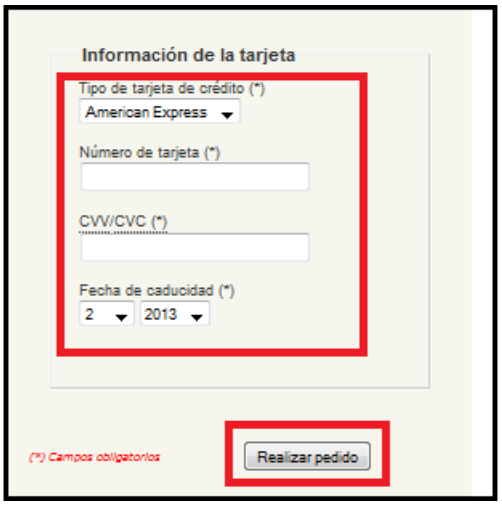


Figura 7.30: Captura del pago mediante tarjeta de crédito

- **Transferencia bancaria / ingreso en efectivo**: Si selecciona está opción le aparece una pantalla con la información necesaria para la realización mediante está forma de pago.

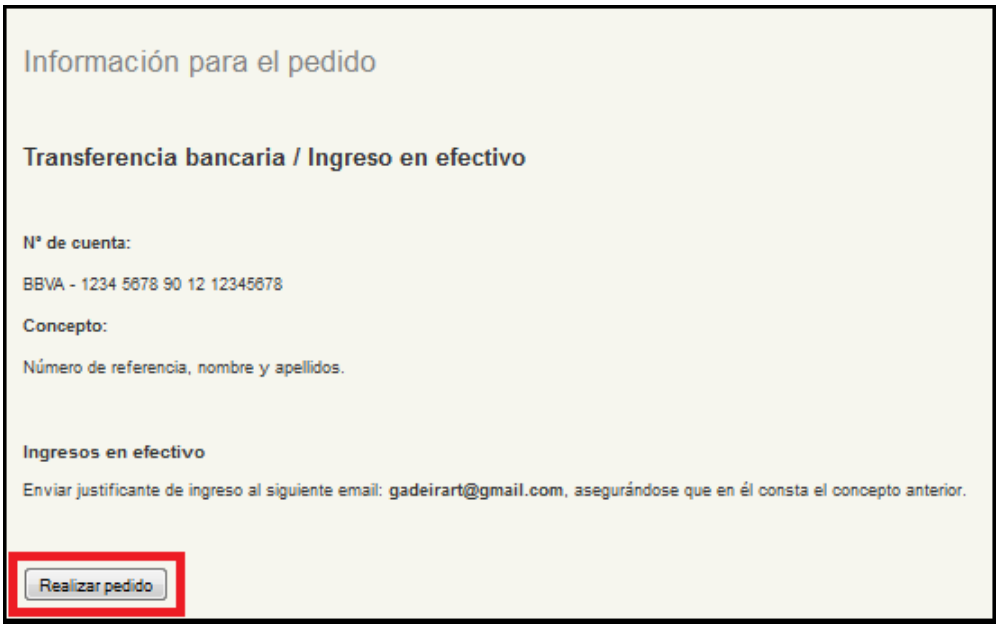


Figura 7.31: Captura del pago mediante transferencia bancaria/ingreso en efectivo

7.4.2.9. Quienes somos

Si deseamos conocer la actividad de la galería tendremos que pulsar sobre el menú *"Quienes somos"* (figura 7.32) que aparece en cualquiera de las páginas de la tienda virtual.

7.4.2.10. Condiciones generales

Para conocer las *condiciones generales* (figura 7.33) de la Galería tendremos que hacer click sobre el menú *"Condiciones "* que aparece en cualquiera de las páginas de la web.

7.4.2.11. Contactar

Obtener la información para poder contactar con la Galería de Arte (figura 7.34), se hace desde el menú *"Contactar"* de la barra de menús.

A través del enlace de *"Google map"* se tendrá acceso al mapa con la localización exacta de la Galería.

Figura 7.32: Captura de quienes somos

Figura 7.33: Captura de las condiciones generales

Figura 7.34: Captura de como contactar con la galería

7.4.2.12. Cambiar de idioma

El *usuario* tendrá la funcionalidad de poder cambiar el idioma de la interfaz web en cualquier momento. Para ello tendrá los siguientes botones en cualquier pantalla.

Figura 7.35: Captura del botón para cambiar la página a inglés

Figura 7.36: Captura del botón para cambiar la página español

Pulsando sobre ellos automáticamente la página web cambiará al idioma seleccionado.

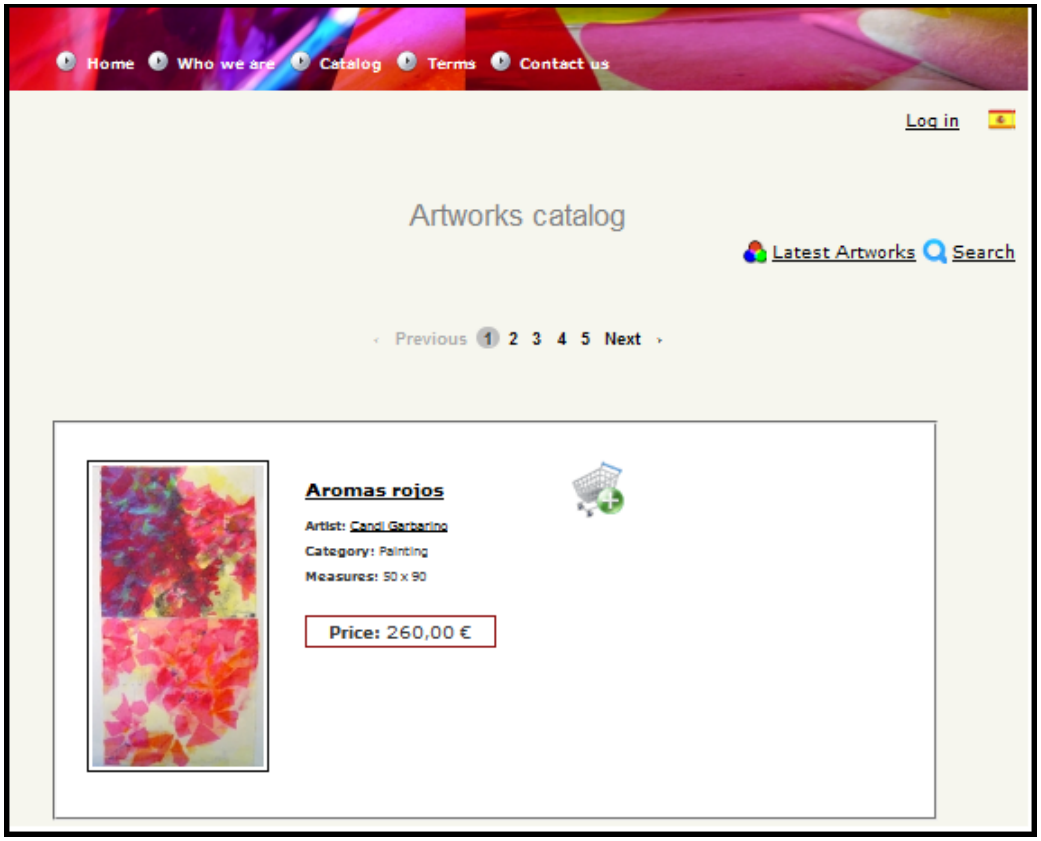


Figura 7.37: Captura del catálogo traducido

Capítulo 8

Manual de instalación y explotación

8.1. Introducción

Este manual proporciona una guía para la *instalación* y *explotación* del sistema desarrollado para la *galería de arte "GadeirArt"*.

8.2. Requisitos previos

Los requerimientos que el sistema debe tener para el correcto funcionamiento. Entre paréntesis pondré la versión con la que yo he trabajado.

- *S.O.*: Ubuntu (Ubuntu Linux 11.04)
- *Lenguajes*: Ruby (v. 1.8.7)
- *Gestor de paquetes*: RubyGems
- *Framework*: Rails (v. 3.0.9)
- *Base de datos*: MySql (MySql Server Versión 5.1.54 - 1ubuntu4)
- *Cliente de subversion*: Svn.

8.3. Inventario de componentes

A continuación se muestran los componentes software que se incluyen en la versión del producto y donde se pueden localizar dentro de la aplicación:

- **Gemas**. La configuración de las gemas se encuentran dentro del fichero *Gemfile*.
- **Modelos**. Carpeta *app\models*.
- **Vistas**. Carpeta *app\views*.
- **Controladores**. Carpeta *app\controllers*.
- **Tests**. Carpeta *test*.

8.4. Procedimientos de instalación

Descarga de la aplicación

Lo primero que tendremos que hacer será bajarnos la última versión del repositorio de nuestra aplicación, para ello ejecutaremos el siguiente comando desde la línea de consola:

```
$>svn co http://subversion.assembla.com/svn/pitig/
```

Instalar librerías adicionales

Como se ha indicado anteriormente en la *sección 8.2 Requisitos previos*, se deberá tener instalados *Rails, Ruby* y *MySql*. Aún así, puede que necesitaremos de ciertas librerías adicionales que permiten un correcto funcionamiento de las gemas. La instalación de las mismas se encuentra recogida en el script ***librerias.sh***.

Para instalarlas nos situamos en el directorio donde se ha descargado la aplicación ***pitig*** y entramos en el subdirectorio ***galeriaArte***, y ejecutamos el script ***librerias.sh***.

```
$>bash librerias.sh
```

```
echo "Instalar OpenSSL para Ruby"
sudo apt-get install libruby

echo "Instalar Librerias adicionales para Nokogiri"
sudo apt-get install libxslt1-dev libxml2-dev
```

```
echo "Instalar Libreria de Java"
sudo apt-get install default-jdk

echo "Instalar Libreria adicional para Paperclip"
sudo apt-get install imagemagick
```

Código 8.1: Contenido del fichero *"librerias.sh"*

Instalar la aplicación

Volvemos a situarnos en el subdirectorio ***galeriaArte*** del directorio ***pitig***, ejecutando posteriormente el script ***instalacion.sh***:

$>bash instalacion.sh

Este realizará los siguientes pasos:

- Instalará las gemas necesarias.
- Eliminará las bases de datos.
- Creará y actualizará las nuevas BBDD (cargándose el fichero de órdenes .sql incluido en instalacion.sh).
- Creará las rutas necesarias.
- Iniciará el servidor.

```
#! /bin/bash
echo "################################"
echo "## Instalacion de Gemas ##"
echo "################################"
export PATH=${PATH}:/var/lib/gems
sudo bundle install

# Necesitamos la contrasena de MySQL
echo "################################"
echo "## Entrar en MySQL      ##"
echo "################################"

# Ahora vamos a eliminar las bases de datos anteriores
```

```
# para posteriormente crearlas y actualizarlas
echo "-> Eliminacion de la BD"
rake db:drop:all

# Leer contrasena de MySQL
echo "Introduzca su contrasena de 'root' para MySQL: "
mysql -u root -p < ./db/ordenes.sql

echo "-> Creando BD"
rake db:create RAILS_ENV=development
echo "-> Actualizando BD"
rake db:migrate RAILS_ENV=development
echo "-> Poblando BD"
rake db:fixtures:load
echo "-> Cargando test"
rake db:test:load

# Crear las rutas
echo "-> Creamos las rutas"
rake routes

# Iniciamos el servidor
echo "-> Iniciando servidor de la aplicacion"
echo "################################"
echo "## Abrir aplicacion     ##"
echo "################################"
firefox http://localhost:3000 &
rails server
```

Código 8.2: Contenido del fichero *"instalacion.sh"*

8.5. Procedimientos de operación y nivel de servicio

A continuación voy a describir algunos procedimiento necesarios para asegurar el correcto funcionamiento de la aplicación en su máquina local.

Configuración del entorno

En *Ruby on Rails* tenemos tres tipos de entornos (ver *sección 5.2 Código fuente*), cada uno de los cuales ayuda a desarrollar las aplicaciones de una forma más cómoda y eficiente, ayudando en el ciclo de desarrollo:

- **Development**: Entorno de desarrollo.
- **Test**: Entorno para pruebas.
- **Production**: Entorno de producción.

Para cambiar de entorno debemos abrir el archivo **config/environment.rb** y reemplazar:

```
RAILS_ENV = ENV['RAILS_ENV'] || 'production'
```

por:

```
RAILS_ENV = 'development'
```

Configuración de las Gemas

Para respetar la configuración existente, en primer lugar compruebe que tiene todas las gemas instaladas.

La siguiente orden se encarga de comprobar las gemas e instalar las que faltan:

```
$>bundle install
```

Configuración de la BD

Las bases de datos con las que puede interactuar la aplicación se especifican en el archivo de configuración **config/database.yml**.

El archivo contiene secciones para los diferentes entornos en los que *Rails* se puede ejecutar de forma predeterminada. Éste trae una configuración de BD predeterminada utilizando *SQLite3*, así que debe de ser modificado ya que se ha elegido como BD *MySQL*.

El fichero de configuración **database.yml** deberá quedar de la siguiente manera:

```
development:
  adapter:  mysql
  database: galeriaArte_development
  username: galeriaArte
  password: hacked
  encoding: utf8
```

```
test:
  adapter:  mysql
  database: galeriaArte_test
  username: galeriaArte
  password: hacked
  encoding: utf8

production:
  adapter:  mysql
  database: galeriaArte_production
  username: galeriaArte
  password: hacked
  encoding: utf8
```

Código 8.3: Contenido del fichero *"database.yml"*

Como se ha indicado anteriormente en la *sección 8.2 Requisitos previos*, deberá tener instalado *MySql* como gestor de BBDD.

Se debe otorgar permisos al usuario *galeriaArte* sobre las bases de datos: *galeriaArte_development* y *galeriaArte_test*. Esta asignación de privilegios se consigue, si el servidor es la máquina local, con las ordenes:

```
mysql>grant all on galeriaArte_development.* to 'galeriaArte'@'localhost'
                        identified by 'hacked';
```

```
mysql>grant all on galeriaArte_test.* to 'galeriaArte'@'localhost'
                        identified by 'hacked';
```

8.6. Pruebas de implantación

Acceso a la aplicación

Desde nuestro navegador preferido, debemos introducir la URL:

```
http://localhost:3000
```

En caso de que el script **instalacion.sh** (descrito en el *párrafo 8.4 Instalar la aplicación*) haya abierto el navegador, es recomendable recargar la página, ya que hay que esperar que se cargue *Rails*.

Vemos como se carga la página de inicio de la aplicación:

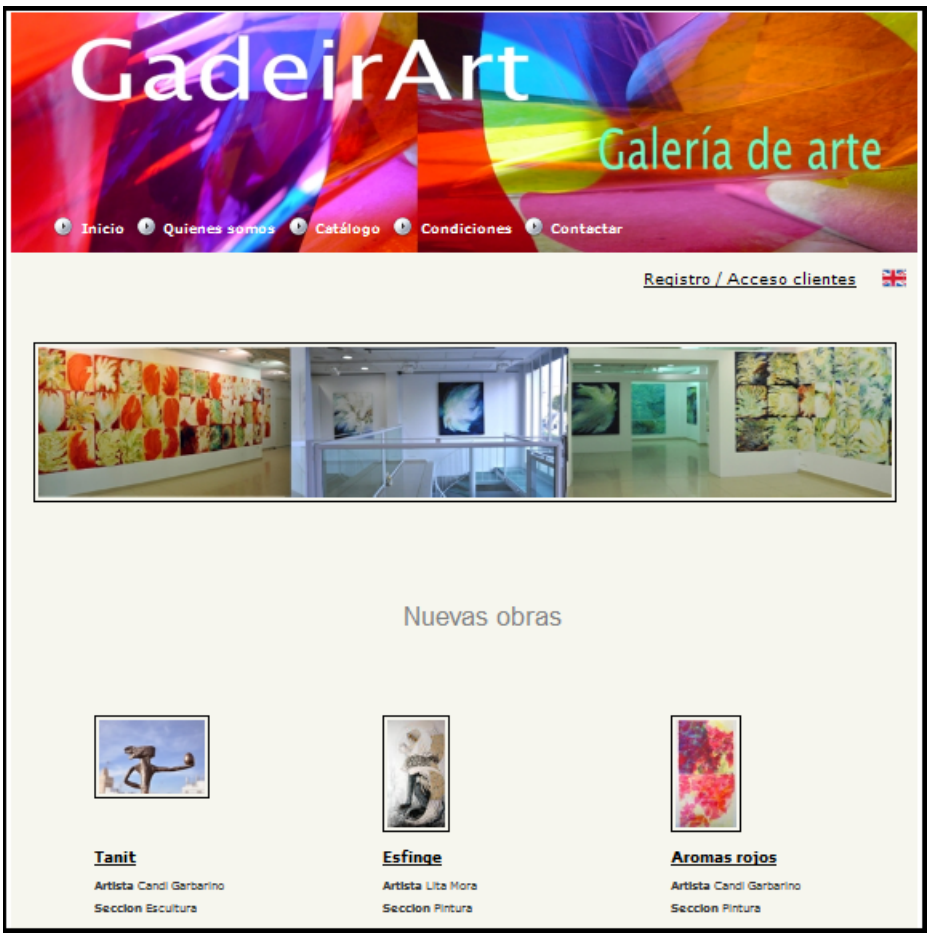


Figura 8.1: Captura del página de inicio

Capítulo 9

Conclusiones

En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre la aplicación desarrollada.

9.1. Objetivos

Los objetivos que me marque al comienzo de este proyecto (*sección 1.2 Propósito, objetivos y alcance del proyecto*) han sido más que satisfactorios, tanto en el plano personal como en el profesional. Quería:

Aprender a desarrollar, implementar y poner en marcha una aplicación Web que pueda tener una utilidad comercial, empleando la herramienta de desarrollo **Ruby on Rails** y los conocimientos adquiridos durante la carrera.

Esto se ha logrado como se puede comprobar a través de:

- las aplicaciones Web desarrolladas utilizando *Ruby on Rails*,
- esta memoria en la que se describe todo el trabajo llevado a cabo para conseguir el desarrollo completo de la aplicación,
- la herramienta que he utilizado para llevar el control del desarrollo y la coordinación de las actividades de mi proyecto, ***Assembla***.

`https://www.assembla.com/spaces/pitig/`

Adaptar la aplicación a distintos requisitos permitiéndome obtener otra a un menor coste. Satisfecho a través de la aplicación generada para el Artista reutilizando gran parte del código de la aplicación de la Galería de Arte.

Realizar un análisis comparativo del framework Ruby on Rails frente a otras alternativas, evaluando las ventajas y características de este respecto a otros. Complacido con el estudio realizado en la memoria en el apartado *3.6 "Estudio de alternativas tecnológicas"*.

La aplicación realizada responde a las expectativas y requerimientos recogidos en la especificación de requisitos en el *capítulo 3*.

9.2. Lecciones aprendidas

En esta sección se detallan las buenas prácticas adquiridas, un análisis de métricas, una comparación cuantitativa del tiempo y el esfuerzo invertido y por último, los problemas encontrados durante el desarrollo del proyecto.

9.2.1. Buenas prácticas

Ha sido muy satisfactorio involucrarme en el proyecto completo, ya que he tenido que desempeñar el papel de varios roles: jefe de proyecto, administrador de BBDD y analista/programador, aprendiendo en cada uno de ellos algo nuevo.

He aprendido a *Planificar y Gestionar* un proyecto, llegando a la conclusión de que es bastante difícil cumplir con unas fechas sino se ha estudiado, analizado y planificado todos sus riesgos.

He puesto en práctica los pequeños conocimientos adquiridos durante el *Taller de LATEX*, impartido por la Escuela, y he obtenido otros muchos realizanco esta memoria.

Durante el desarrollo he aprendido a programar utilizando un nuevo entorno de desarrollo *Ruby on Rails*. Esto me ha llevado bastante tiempo al principio, ya que he tenido que aprender su funcionamiento y su estructura para poder comenzar, pero luego he ido aprendiendo a la vez que iba desarrollando la aplicación.

Ahora puedo destacar lo cómodo y sencillo que resulta el adaptarse a este framework, el cual facilita y acelera la función del desarrollador mediante unas configuraciones de inicio muy efectivas, una estructura muy clara y, sobre todo, el uso de un lenguaje que resulta realmente intuitivo y funcional como es *Ruby*.

9.2.2. Métricas

En esta sección voy a mostrar las métricas realizadas sobre la aplicación.

9.2.2.1. Métricas del producto

A continuación podrá visualizar algunas de las estadísticas del sistema software a través de la aplicación **StatsSVN** (encontrará más en la **carpeta "estadisticas_svn"** contenida en el *CD adjunto*) y el análisis de estadísticas de código incluido dentro de *Ruby On Rails*.

En la *figuras 9.1, 9.2, 9.5, 9.4, 9.3 y 9.6* podemos ver las estadísticas de código clasificadas según extensión y ordenados por el número de líneas de los directorios principales de la aplicación.

File Types

Type	Files	LOC	LOC per file
*.erb	157 (65.1%)	3701 (54.6%)	23.5
*.rb	69 (28.6%)	3062 (45.1%)	44.3
*.rxml	1 (0.4%)	20 (0.3%)	20.0
Non-Code Files	14 (5.8%)	0 (0.0%)	0.0
Totals	241 (100.0%)	6783 (100.0%)	28.1

Figura 9.1: Estadísticas del repositorio del directorio "app"

File Types

Type	Files	LOC	LOC per file
*.yml	6 (27.3%)	1560 (76.2%)	260.0
*.rb	15 (68.2%)	486 (23.8%)	32.4
Non-Code Files	1 (4.5%)	0 (0.0%)	0.0
Totals	22 (100.0%)	2046 (100.0%)	93.0

Figura 9.2: Estadísticas del repositorio del directorio "config"

File Types

Type	Files	LOC	LOC per file
*.js	11 (12.9%)	31458 (95.9%)	2859.8
*.css	12 (14.1%)	1261 (3.8%)	105.0
*.html	3 (3.5%)	78 (0.2%)	26.0
*.txt	1 (1.2%)	5 (0.0%)	5.0
Non-Code Files	58 (68.2%)	0 (0.0%)	0.0
Totals	85 (100.0%)	32802 (100.0%)	385.9

Figura 9.3: Estadísticas del repositorio del directorio "public"

File Types

Type	Files	LOC	LOC per file
*.rb	2 (66.7%)	48 (100.0%)	24.0
Non-Code Files	1 (33.3%)	0 (0.0%)	0.0
Totals	3 (100.0%)	48 (100.0%)	16.0

Figura 9.4: Estadísticas del repositorio del directorio "lib"

File Types

Type	Files	LOC	LOC per file
*.rb	22 (95.7%)	673 (68.1%)	30.5
*.sql	1 (4.3%)	315 (31.9%)	315.0
Totals	23 (100.0%)	988 (100.0%)	42.9

Figura 9.5: Estadísticas del repositorio del directorio "db"

File Types

Type	Files	LOC	LOC per file
*.rb	85 (78.0%)	5217 (78.2%)	61.3
*.yml	22 (20.2%)	1453 (21.8%)	66.0
Non-Code Files	2 (1.8%)	0 (0.0%)	0.0
Totals	109 (100.0%)	6670 (100.0%)	61.1

Figura 9.6: Estadísticas del repositorio del directorio "test"

En la *figuras 9.7* podemos ver el análisis del código fuente según la estructura del proyecto realizado en *Ruby On Rails* [Fowler, Martin].

```
+----------------------+-------+-------+---------+---------+-----+-------+
| Name                 | Lines |   LOC | Classes | Methods | M/C | LOC/M |
+----------------------+-------+-------+---------+---------+-----+-------+
| Controllers          |  2301 |  1493 |      26 |     186 |   7 |     6 |
| Helpers              |    67 |    62 |       0 |       2 |   0 |    29 |
| Models               |   640 |   320 |      15 |      24 |   1 |    11 |
| Libraries            |    48 |    29 |       1 |       7 |   7 |     2 |
| Integration tests    |  1460 |   893 |      17 |      79 |   4 |     9 |
| Functional tests     |  1712 |   900 |      26 |       0 |   0 |     0 |
| Unit tests           |  2024 |  1170 |      40 |       1 |   0 |  1168 |
+----------------------+-------+-------+---------+---------+-----+-------+
| Total                |  8252 |  4867 |     125 |     299 |   2 |    14 |
+----------------------+-------+-------+---------+---------+-----+-------+
  Code LOC: 1904     Test LOC: 2963     Code to Test Ratio: 1:1.6
```

Figura 9.7: Estadísticas del código fuente Rails

A continuación pasamos a describir los términos que aparecen en la *figuras 9.7* para una mejor interpretación de la misma:

- **Lines**: Líneas de código totales de cada una de las categorías.
- **LOC**: Líneas de código reales no autogeneradas por el framework.
- **Classes**: Número de clases pertenecientes a cada una de las categorías.
- **Methods**: Número de métodos correspondientes a cada una de las categorías.
- **M/C**: Media de métodos por clase.
- **LOC/M**: Media de número de líneas por método.

A través de estas estadísticas se deduce que es mucho más factible programar utilizando el framework que hacer la aplicación desde el comienzo sin éste, ya que se ahorra tiempo y líneas de código que nos facilita *Ruby on Rails*.

9.2.2.2. Métrica del proceso

A continuación se recoge una comparación cuantitativa del tiempo y el esfuerzo realmente invertido frente al estimado y planificado. Estos datos han sido recogidos a través de *Assembla* (`https://www.assembla.com/spaces/pitig/`), el sistema de gestión de tareas empleado para el seguimiento del presente proyecto.

He tenido en cuenta los siguientes parámetros:

- *Días a la semana:* 5
- *Horas que puedo invertir por día:* 4h.

A continuación, en la *figura 9.8*, muestro el total de **horas invertidas en el proyecto y desarrollo de la presente memoria**, equivalente a **60 semanas** (20h/semana).

En la *imagen 9.9*, en el *Sprint 14*, podemos ver el total de **horas invertidas en el desarrollo de la presente memoria** equivalente, aproximadamente, a **20 semanas** (20h/semana).

Visto lo anterior, podemos observar que he empleado en el **desarrollo del proyecto** 808,50h. equivalentes, aproximadamente, a **40 semanas** de trabajo (20h/semana). Estas horas se han sacado descontando el nº de horas invertidas en la memoria *(imagen **??**)* del nº de horas invertidas en total: proyecto y memoria *(imagen 9.8)*.

En la *figura 9.9* muestro, mas detalladamente, las horas estimadas e invertidas por hitos, junto con el nº de tickets empleados en cada uno de los sprints y en la *figura 9.10* muestro su respectiva gráfica.

Lista de Tareas desde el 2011-07-01 hasta el 2013-04-02 Descargar c

Fecha	Usuario	Horas	Descripción
		1207,55	
2013-04-02	Silvia María Garbarino de la Rosa	3,00	'Memoria' - Resumen
2013-04-02	Silvia María Garbarino de la Rosa	0,30	'Asembla' - Actualización
2013-04-01	Silvia María Garbarino de la Rosa	5,25	'Memoria' - Resumen

Figura 9.8: Total de horas invertidas en el proyecto y memoria (Assembla)

Se observa que las **horas estimadas para la realización del proyecto** son 1.440h, equivalentes, aproximadamente, a **72 semanas** de trabajo (20h/semana).

A continuación muestro el número de *horas invertidas cada mes en el desarrollo del proyecto y de la memoria.*

Horas invertidas	
Mes	**Horas**
jul-11	8,9
ago-11	53,2
sep-11	103,95
oct-11	63,43
nov-11	119,06
dic-11	81,22
ene-12	52,49
feb-12	74,88
mar-12	79,65
abr-12	116,53
may-12	52,99
jun-12	90,13
jul-12	71,11
ago-12	1,95
sep-12	19,3
oct-12	22,85
nov-12	24,2
dic-12	43,43
ene-13	68,28
feb-13	36,35
mar-13	12,8
abr-13	10,85
Total horas	**1183,9**
Nº semanas	**30 (40h/semana)**

Figura 9.11: Horas invertidas cada mes (Assembla)

Nº	Sprint	Nº de tickets	Horas estimadas	Horas invertidas
0	Sprint	8	224	63,42
1	Comienzo aplicación	8	24	24,13
2	Artistas	5	40	30,23
3	Colecciones	4	20	15,77
4	Obras	5	80	68,63
5	Secciones	4	20	12,33
6	Técnicas	4	20	17,87
7	Catálogo	5	120	123,5
8	Despliegue	6	60	93,04
9	Cesta de la compra	2	40	45,42
10	Etiquetado	4	28	26,12
11	Autenticación	5	80	71,77
12	Compra y procesamiento de pedidos	10	120	130,19
13	Internacionalización	15	80	85,28
14	Memoria	12	400	351,32
15	Menú Inicio	1	8	6,09
16	Menú Contactar	2	8	3
17	Menú Quienes Somos	6	8	7,5
18	Menú Condiciones Generales	7	8	6,25
19	Menú Administración	1	12	13
20	Presentación proyecto	1	40	0,7
	Total	**115**	**1.440,00**	**1.195,56**

Figura 9.9: Horas estimadas e invertidas por hitos (Assembla)

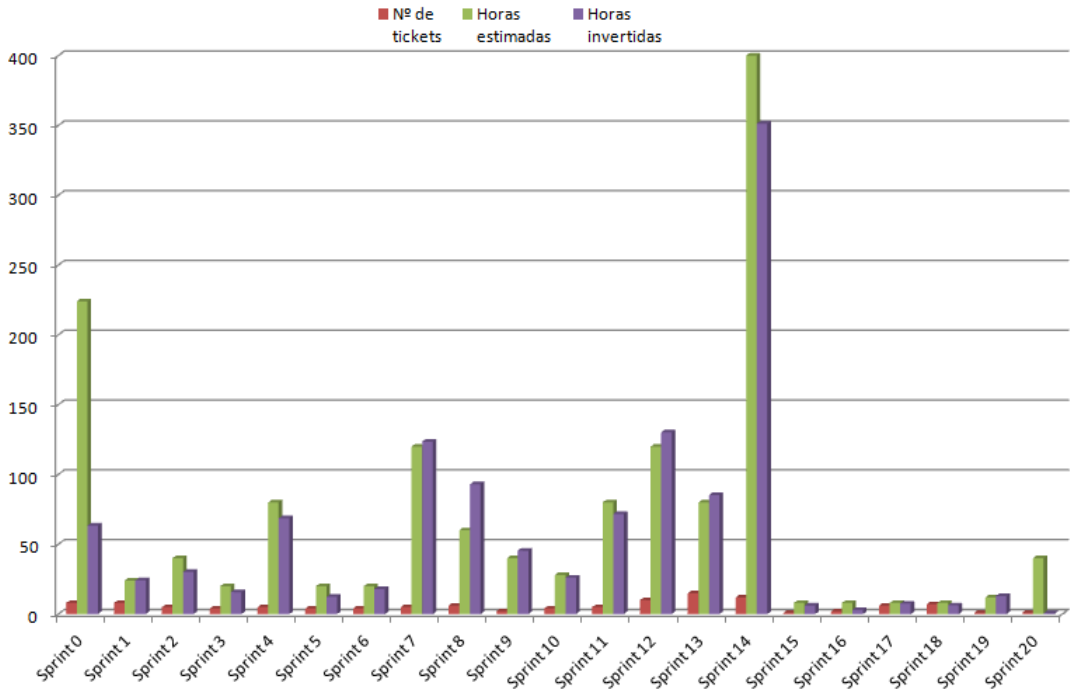


Figura 9.10: Gráfica de horas por hitos (Assembla)

El esfuerzo realizado durante el proyecto también está recogido en la carpeta **changelog** del CD adjunto, la cual contiene el trabajo que he ido desarrollando cada día, es decir, los cambios ejecutados en el proyecto.

```
19/abril/2012
  * Obras - Controlador - Destroy
  * Colecciones - Controlador - Destroy -
  Modificación para que no se borre si tiene obras asociadas
  * Secciones - Controlador - Destroy -
  Modificación para que no se borre si tiene obras u
  técnicas asociadas
  * Tecnicas - Controlador - Destroy -
  Modificación para que no se borre si tiene obras asociadas
  * Artistas - Controlador - Destroy -
  Modificación para que no se borre si tiene obras asociadas
  * Actualización changelog
  * Memoria - Bibliografía
  * Memoria - Capítulo 4 - Implementación - Gemas -
  MetaSearch
  * Memoria - Capítulo 4 - Análisis - Historias de usuarios -
  Obras y Catálogo
  * Catalogo - Busqueda con gema metasearch
  * Obras - Busqueda con gema metasearch
  * Catalogo - Eliminación de lo relacionado con gema sunspot
  * Memoria - Capítulo 4 - Implementación - Gemas -
  Eliminación sunspot
  * Catalogo - Búsqueda - Añadido botón limpiar formulario
  * Obras - Búsqueda - Añadido botón limpiar formulario
  * Búsqueda de información sobre contenedores con pestañas
  en Rails

18/abril/2012
  * Catalogo - Recomendaciones - Modificaciones
  * Obras - Busqueda con gema metasearch
  * Obras - Gema meta_search - Investigación
  * Obras, artistas, colecciones, secciones, técnicas -
    Modificación botón 'new' en index
  * Secciones - Integridad referencial
```

Código 9.1: Extracto del contenido del fichero Changelog

Resumiendo, puedo comentar que he estimado más o menos el número de semanas necesarias para la realización del proyecto y de la memoria. Teniendo en cuenta, que en este estudio están contabilizadas las 10 semanas (20h/semana) que invertiré próximamente para la realización de la presentación del proyecto.

	(20h/semana)	(40h/semana)
Semanas estimadas	72 semanas	36 semanas
Semanas invertidas	60 semanas	30 semanas
Desviación temporal	12 semanas	6 semanas

Tabla 9.1: Estudio del tiempo del proyecto

Para concluir, la experiencia ha sido muy grata junto con un aprendizaje muy completo y satisfactorio.

9.2.2.3. Amortización del desarrollo de la aplicación

A partir de ahora, una vez realizada la primera aplicación, puedo generar cualquier aplicación web a medida para galerías y/o artistas con ***un coste menor***, amortizando el importe de la realización de la aplicación para la Galería de Arte.

Para mostrarlo he desarrollado una segunda aplicación, consistente en una *Tienda Virtual para un Artista* (`http://artista.herokuapp.com/`), adaptando la realizada para la *Galería de Arte* a nuevos requisitos.

Con esto quiero demostrar la ventaja de construir una nueva aplicación, partiendo y reutilizando el código de otra, respecto al coste y el tiempo empleado inicializándola desde cero.

A continuación se recoge una comparación cuantitativa del tiempo y el esfuerzo invertido en la aplicación. Estos datos también han sido recogidos a través del sistema de gestión *Assembla* (`https://www.assembla.com/spaces/pitig_artista/`).

He tenido en cuenta, al igual que antes, los siguientes parámetros:

- *Días a la semana:* 5
- *Horas que puedo invertir por día:* 4h.

En la imagen 9.12 muestro el total de **horas invertidas en la aplicación para el Artista**.

Fecha	Usuario	Horas	Descripción
		42,23	
2013-03-21	Silvia María Garbarino de la Rosa	1,00	Pruebas de integración
2013-03-20	Silvia María Garbarino de la Rosa	1,00	Pruebas unitarias
2013-03-20	Silvia María Garbarino de la Rosa	1,25	Pruebas funcionales
2013-03-20	Silvia María Garbarino de la Rosa	4,50	Modificaciones varias
2013-03-19	Silvia María Garbarino de la Rosa	2,80	Biografia artista, pagina inicio

Figura 9.12: Total horas invertidas en la aplicación para el artista (Assembla)

He reutilizado la mayoría del código de la aplicación desarrollada para la *Galería de Arte*: controladores, vistas, modelos . . . ya que la aplicación viene a ser similar, cambiando tan solo unos cuantos requisitos. Por ejemplo, he creado un nuevo modelo para las exposiciones del *Artista* junto con su controlador y sus vistas; se ha cambiado la vista del catálogo; en lugar de mostrarse el apartado "Quienes somos" de la *Galería* se muestra el curriculum de la *Artista*, con lo que este código se ha ampliado para satisfacer este requisito, modificándose también con éste la página de inicio de la aplicación; se ha quitado el apartado de contactar y algunas otras pequeñas modificaciones . . .

Horas invertidas para:

- *Web de la Artista*: **2 semanas** (20h/semana), equivalentes a **1 semana** (40h/semana)

  frente a

- *Web de la Galería*: **30 semanas**(40h/semana), equivalentes aproximadamente a **7 meses**

Podemos comprobar que he ganado bastante reutilizando gran parte del código de la *Galería de Arte* para la nueva aplicación, ya que tanto el coste como el tiempo empleado es menor. De esta forma, cada vez que cree una aplicación para una Galería y/o Artista iré amortizando el coste de la primera aplicación desarrollada.

Recopilando, tenemos en cuenta los siguientes parámetros para el cálculo de la segunda aplicación:

- *Personas implicadas en el proyecto:* 1
- *Días a la semana:* 5
- *Horas invertidas por día:* 8h.
- *Semanas empleadas:* 1 semana.

A continuación, podemos ver el coste de la aplicación para el artista recogida en la siguiente tabla:

Nombre	**Descripción**	**Tipo**	**Coste**	**Total**
Personal	Personal cualificado y especializado	Humano	2.375€/mes	593,75€/semana
Equipo informático	PC e impresora	Activo	42€/mes	10,5€/semana
Costes indirectos	Tinta, luz...	Material	10 % del coste del personal	59,37€/semana
			Total	**663,62€**

Tabla 9.2: Costes del proyecto para la aplicación del Artista

Comparativa del coste entre ambas aplicaciones (ver sección *Costes* 2.5):

Nombre	**Coste**	**Total Galería** (30 semanas)	**Total Artista** (1 semana)
Personal	2.375€/mes	16.625€	593,75€/semana
Equipo	42€/mes	294€	10,5€/semana
Costes	10 % del personal	1.662,5€	59,37€/semana
	Total	**18.581,50€**	**663,62€**

Tabla 9.3: Comparativa de costes de ambas aplicaciones

9.2.2.4. Estudio de la usabilidad de la aplicación

Para medir la usabilidad de la aplicación contaré con la colaboración de varios usuarios, entre ellos con la galerista y voy a tener en cuenta las siguientes variables:

- **Efectividad**: Me permite medir la exactitud con la que se alcanzan los objetivos de una tarea concreta. En este caso voy a medir el *porcentaje total de tareas completadas* por el usuario y la *dificultad en realizarla.*
- **Eficiencia**: Se refiere al esfuerzo que un usuario tiene que hacer para conseguir un objetivo. Mediré el *tiempo empleado* en completar cada tarea.
- **Satisfacción**: Mediré el *grado de satisfacción* de los usuarios.

Tras realizar a los usuarios un pequeño test *(Anexo I)* con siete tareas, he obtenido los siguientes resultados reflejados en los siguientes gráficos. Los resultados de los test podemos visualizarlos en la **carpeta "usabilidad"** contenida en el *CD adjunto.*

Todos los usuarios han conseguido completar las diferentes tareas que se les ha recomendado en el test de usabilidad.

La media de dificultad con que se han encontrado los usuarios al realizar cada una de las tareas la podemos ver en el gráfico 9.13.

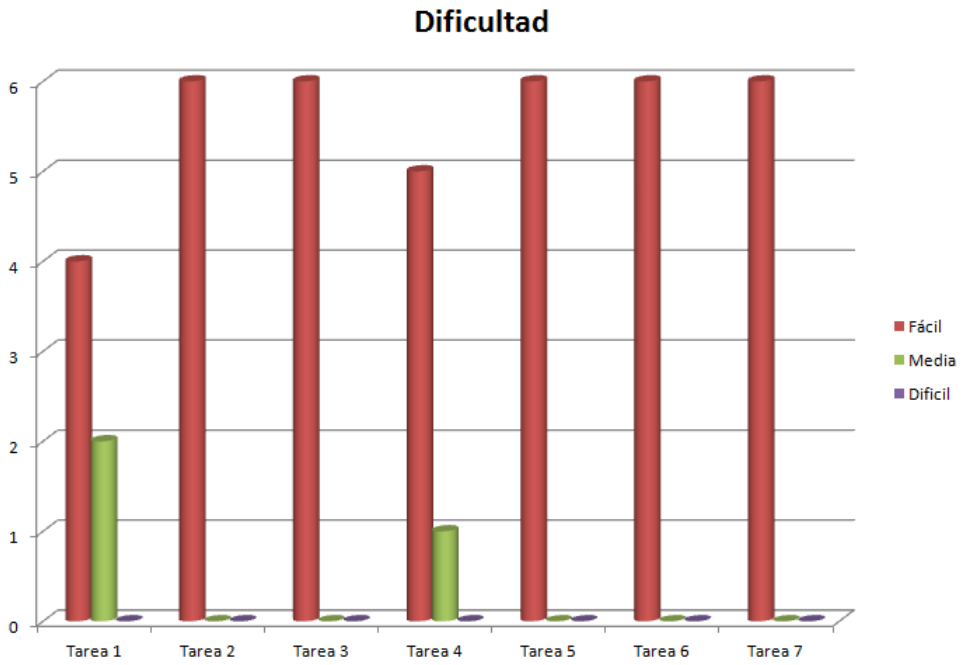


Figura 9.13: Test usabilidad - Dificultad

La gráfica 9.14 muestra la media de tiempo (expresada en minutos) empleado por todos los usuarios para completar cada una de las 7 tareas.

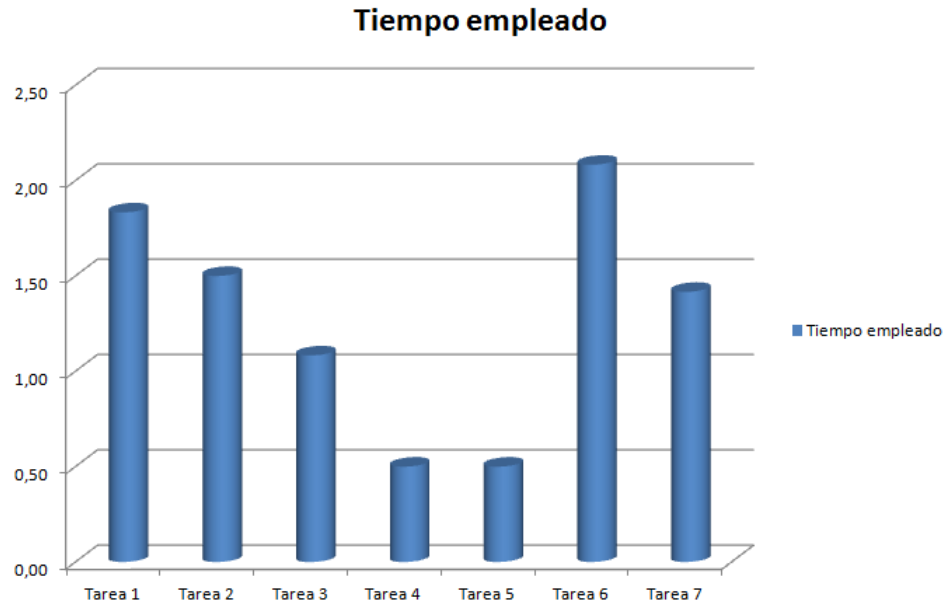


Figura 9.14: Test usabilidad - Tiempo empleado

A continuación, en el gráfico 9.15 vemos reflejado la media de satisfacción de los usuarios en cada una de las tareas realizadas.

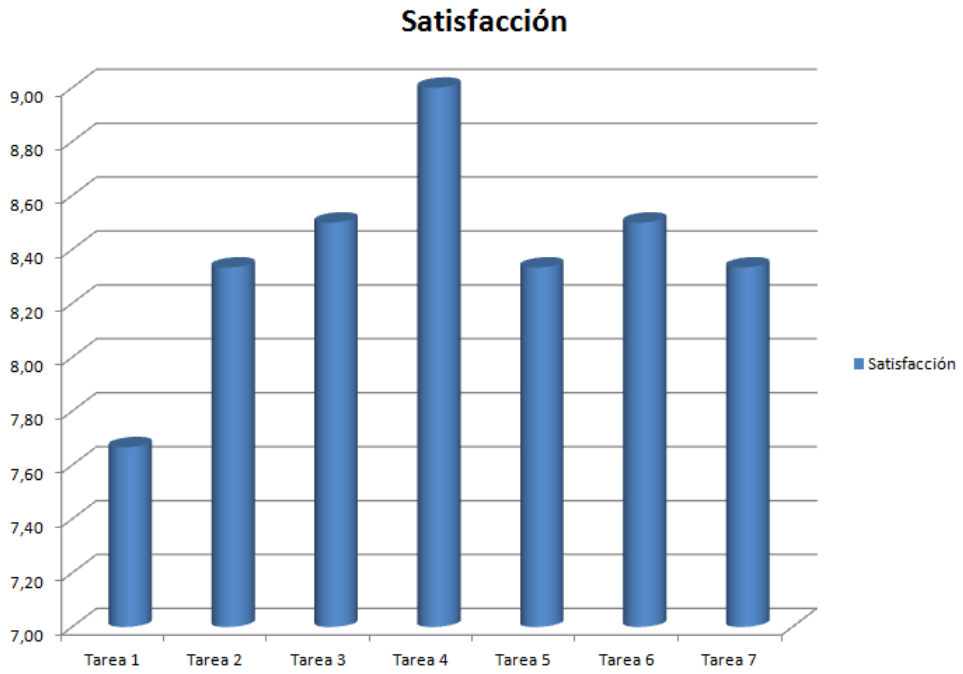


Figura 9.15: Test usabilidad - Satisfacción

9.2.3. Problemas encontrados

Voy a destacar algunos de ellos aunque, durante todo el desarrollo, me he ido encontrando con problemas que he ido solventando poco a poco.

Uno de los problemas que me encontré fue que tras tener implementada la búsqueda de obras de arte a través de la *gema Sunspot*, la cual me llevo bastante tiempo estudiar como funcionaba y obtener su funcionamiento en mi aplicación, me encontré que al realizar el despliegue en *Heroku*[1] no me permitía utilizar esta gema ya que para poder hacerlo había que pagar por no ser de libre uso. Para solucionarlo opte por usar otra gema *MetaSearch*, la cual también me resulto bastante útil.

Otro problema similar fue el espacio donde guardar las imágenes de los artistas y las obras de arte de la Galería, también en *Heroku* había que pagar por este servicio de alojamiento, así que finalmente pude solucionar el problema utilizando el servicio *dropbox* para alojar las imágenes.

Lo que no he conseguido solventar ha sido el poder añadir las tasas correspondientes al porte del pedido, ya que quise hacerlo a través de la empresa *UPS*[2] y tras tenerlo

[1] *Heroku*: Servicio de Hosting en la nube
[2] *UPS*: Compañía de transporte

implementado a falta de unos datos que me tenía que facilitar la empresa y ponerme en contacto con ellos, no podían facilitarmelos porque tenía que ser cliente de la empresa. Así que lo he dejado como ampliación para un trabajo futuro donde buscaría otra forma de poder facilitar automáticamente las tasas al cliente.

También me encontrado con warnings provenientes de las gemas utilizadas, en donde se advierte que se está utilizando una orden que ya es obsoleta en las nuevas versiones y que habría que sustituir. Para solventarlo habría que estudiar donde ocurre en las gemas y eso no está contemplado dentro del proyecto, ya que llevaría bastante tiempo debido a que las gemas están programadas por otros.

Finalmente, he tenido que solventar algunas vulnerabilidades en heroku, lo cual me ha llevado un tiempo curioso ya que me ha bloqueado hasta la aplicación.

9.3. Trabajo futuro

Para futuras ampliaciones se podría realizar:

- La *implementación del seguimiento del pedido* por parte del cliente.
- El *pago a través de una cuenta paypal* del propio usuario.
- Implementar las *tasas correspondientes al porte del pedido*.

Capítulo 10

Anexos

A continuación se recogen cada uno de los anexos a la presente memoria de mi Proyecto de Fin de Carrera.

A. Acrónimos

AIR *(«Adobe Integrated Runtime»).*
AJAX JavaScript asíncrono y XML *(«Asynchronous JavaScript And XML»).*
API Interfaz de Programación de Aplicaciones *(«Application Programming Interface»).*
B2C Del negocio al consumidor *(«Business-to-Consumer»).*
BD Base de Datos *(«DataBase»).*
BBDD Bases de Datos *(«DataBases»).*
CRUD Crear, Obtener, Actualizar y Borrar *(«Create, Read, Update and Delete»).*
CSS Hojas de Estilo en Cascada *(«Cascading Style Sheets»).*
DOM Modelo de Objetos del Documento *(«Document Object Model»).*
DRY No te repitas *(«Don't Repeat Yourself»).*
DSL Lenguaje Específico de Dominio *(«Domain-Specific Language»).*
E/R Entidad/Relación *(«Entity relationship»).*
EER Entidad-Relación Mejorado *(«Enhanced Entity-Relationship»).*
GNU GNU No es Unix *(«GNU is Not Unix»).*
GPL Licencia Pública General de GNU *(«GNU General Public License»).*
HTML Lenguaje de Marcado de HiperTexto *(«HyperText Markup Language»).*
HTTP Protocolo de Transferencia de Hipertexto *(«Hypertext Transfer Protocol»).*
i18n Internacionalización *(«Internationalization»).*
IEEE Instituto de Ingenieros Eléctricos y Electrónicos *(«Institute of Electrical and Electronics Engineers»).*
JSP *(«JavaServer Pages»).*
MIT Instituto Tecnológico de Massachusetts *(«Massachusetts Institute of Technology»).*
MVC Modelo-Vista-Controlador *(«Model-View-Controller»).*
ORM Mapeo Objeto-Relacional *(« Object-Relational Mapping»).*
PDF Formato de Documento Portátil *(«Portable Document Format»).*
PERT Programa o Proyecto de Evaluación y Revisión Técnica *(«Program (or Project) Evaluation and Review Technique»).*
PHP *(«Hypertext Pre-processor»).*
png *(«Portable Network Graphics»).*
POO Programación Orientada a Objetos.
RBS Estructura de desglose de recursos *(«Resource Breakdown Structure»).*
RoR Ruby on Rails.
SGBD Sistema de Gestión de Base de Datos.
SMTP Protocolo Simple de Transferencia de Correo *(«Simple Mail Transfer Protocol»).*
SQL Lenguaje de Consulta Estructurado *(«Structured Query Language »).*
TCP/IP Protocolo de Control de Transmisión *(«Transmission Control Protocol»)*/Protocolo de Internet *(«Internet Protocol»).*
TDD Desarrollo Guiado por Pruebas *(«Test-Driven Development»).*
UTF-8 *(«8-bit Unicode Transformation Format»).*

VPS Servidor Privado Virtual *(«Virtual Private Servers»)*.
W3C *(«World Wide Web Consortium»)*.
WBS Estructura de Desglose del Trabajo *(«Work Breakdown Structure»)*.
XML Lenguaje de Marcas eXtensible *(«eXtensible Markup Language»)*.
YAML YAML no es otro lenguaje de marcado *(«YAML Ain't Markup Language»)*.

B. Definiciones

ActiveRecord Patrón de diseño que permite crear un objeto que *envuelve* una tabla SQL, agregándole la *lógica del modelo* y el *control de acceso*. Este patrón permite unir el mundo de la programación orientada a objetos (POO), que es un mundo intuitivo y natural, con el mundo matemático y rígido de los datos relacionales (SQL).

Administrador del Sistema Persona que tiene la responsabilidad de ejecutar, mantener, operar y asegurar el correcto funcionamiento de la aplicación.

AJAX Técnica de desarrollo web para crear aplicaciones interactivas.

Andamiaje *(«Scaffold»)* es poder realizar todas las operaciones que se nos pide con tan solo definir la entidad.

API Conjunto de funciones y procedimientos (o métodos, en POO) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Assembla Es una herramienta web que permite el control de desarrollo y la coordinación de las actividades de un proyecto.

B2C Estrategia que desarrollan las empresas comerciales para llegar directamente al cliente o usuario final. En la práctica, suele referirse a las plataformas virtuales utilizadas en el comercio electrónico para comunicar empresas (vendedoras) con particulares (compradores). Por eso, el uso más frecuente es *Comercio electrónico B2C*.

Base de datos Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Changelog Archivo que lista los cambios hechos a un proyecto informático desde su última versión.

Comercio electrónico También conocido como *e-commerce («electronic commerce»)*, consiste en la compra y venta de productos o de servicios a través de medios electrónicos, tales como Internet y otras redes informáticas.

Control de versiones Gestión de los cambios que se realizan sobre los elementos de un producto facilitando la administración de las mismas.

CRUD Hace referencia a las funciones básicas en bases de datos: crear, obtener, actualizar y borrar

CSS Tecnología que permite crear páginas web con un diseño más exacto, añadiendo mayores posibilidades a HTML y permitiendo una mayor separación entre la información y la presentación.

El *W3C* es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

Dia Editor de diagramas con las herramientas necesarias para crear o modificar gráficos.

Diagrama de Gantt Herramienta gráfica cuyo objetivo es mostrar, para la planificación del desarrollo de un proyecto, el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

Diagrama PERT Este diagrama es una representación gráfica de las relaciones entre las tareas del proyecto que permite calcular los tiempos de éste de forma sencilla.

DOM Interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

Dropbox Servicio de alojamiento de archivos multiplataforma, operado por la compañía Dropbox. Este servicio permite almacenar y sincronizar archivos en línea y entre computadoras, y compartir archivos y carpetas con otros.

DSL Consiste en proporcionar a los clientes un lenguaje con una sintaxis y una semántica más conveniente para describir la solución al tipo de problema al que se enfrentan.

Framework Conjunto de componentes que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web.

Forja Plataforma de desarrollo colaborativo de software enfocado hacia la cooperación entre desarrolladores para la difusión de software y el soporte al usuario.

Gemas Plugins y/o códigos añadidos a nuestros proyectos RoR, que nos permiten nuevas funcionalidades como nuevos create, nuevas funciones pre-escritas o nuevas herramientas para el desarrollo.

GitHub Forja para alojar proyectos utilizando el sistema de control de versiones Git.

GNU Proyecto que promociona el desarrollo colaborativo de software y conocimiento mediante el uso de licencias libres.

Helpers Módulo que ayuda a las vistas definiendo funciones que devuelven código HTML.

Heroku Plataforma de hosting en la nube orientada a Rails.

Hipertexto Texto que en la pantalla de un dispositivo electrónico, permite conducir a otros textos relacionados, pulsando con el ratón en ciertas zonas sensibles y destacadas. La forma más habitual de hipertexto es la de hipervínculos que van a otros documentos.

Hosting en la nube El alojamiento web en la nube se asienta sobre una red de servidores vinculados para formar una sola plataforma. Debido a que hay una gran cantidad de servidores trabajando en conjunto en lugar de uno solo, se equilibra la carga, aumenta la capacidad, y se reduce al mínimo la probabilidad de fallo.

HTML Lenguaje sencillo de hipertexto, es decir, un lenguaje que permite escribir texto de forma estructurada, y que está compuesto por etiquetas, que marcan el inicio y el fin de cada elemento del documento.

Con el hipertexto lo que se consigue realmente es especificar tanto la estructura lógica del contenido como los diferentes aspectos que se desean dar a dicho contenido. Sin embargo, el uso de *CSS* nos permite separar ambos conceptos: *estructura* y *aspecto visual*.

HTTP Protocolo usado en cada transacción de la *World Wide Web*.

IEEE Asociación técnico-profesional mundial dedicada a la estandarización.

ImageMagick Librería muy potente para realizar manipulaciones de imágenes.

Ingeniería del Software Aquella que ofrece métodos y técnicas para desarrollar y mantener software de calidad.

ISO 3166 Estándar que codifica los nombres de países y áreas dependientes así como sus principales subdivisiones.

Java Lenguaje de programación orientado a objetos, desarrollado a principios de los años 90.

jQuery Biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

JSP Tecnología *Java* que permite generar contenido dinámico para web, en forma de documentos *HTML, XML* o de otro tipo.

LaTeX Sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas.

Lógica de programación Base sobre la cual se sustenta la programación en si.

Máquina virtual Software que emula a una computadora y puede ejecutar programas como si fuese una computadora real.

Metaprogramación Consiste en escribir programas que escriben o manipulan otros programas (o a sí mismos) como datos, o que hacen en tiempo de compilación parte del trabajo que, de otra forma, se haría en tiempo de ejecución.

Metodología ágil Marco de trabajo conceptual de la Ingeniería de Software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto.

Metodología de desarrollo de software Marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

Métrica del producto Medidas de producto Software durante cualquier fase de su desarrollo desde los requisitos hasta la instalación.

Métrica del proceso Medidas del proceso de desarrollo del Software tales como tiempo de desarrollo total, esfuerzo en días/hombre o mes/hombre de desarrollo del producto, tipo de metodología utilizada o nivel medio de experiencia de los programadores.

MIT Licencia que permite reutilizar el software así licenciado tanto para ser software libre como para ser software no libre, permitiendo no liberar los cambios realizados al programa original.

Modelo de negocio Planificación que realiza una empresa respecto a los ingresos y beneficios que intenta obtener. En un modelo de negocio, se establecen las pautas a seguir para atraer clientes, definir ofertas de producto e implementar estrategias publicitarias, entre muchas otras cuestiones vinculadas a la configuración de los recursos de la compañía.

MVC Patrón arquitectónico desarrollado para interfaces gráficas que resalta la importancia de una separación clara entre la presentación de datos y la lógica de negocio de una aplicación.

MySql Sistema de gestión de bases de datos relacional, multihilo y multiusuario.

Navegador web Aplicación que opera a través de Internet, interpretando la información de archivos y sitios web para que podamos ser capaces de leerla.

Open Source Calificación de software que cumple una serie de requisitos, principalmente aquel que permite una libre redistribución, distribuye el código fuente, y permite modificaciones y trabajos derivados.

ORM Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia, que en *Rails* es ActiveRecord.

Patrones de diseño La base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

png Formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps.

PHP Es un lenguaje de programación libre, multiplataforma e interpretado, diseñado para la creación de páginas web dinámicas.

POO Paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento.

Rails Framework de aplicaciones web de código abierto, multiplataforma, distribuido bajo la licencia del MIT.

Rake Equivalente a *make* para *Ruby*. Sirve para crear y automatizar tareas de mantenimiento.

Refactorización Técnica de la Ingeniería de Software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

Reglas de negocio Describen las políticas, normas, operaciones, definiciones y restricciones presentes en una organización y que son de vital importancia para alcanzar los objetivos.

Repositorio Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

REST Conjunto de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes.

RESTful Sistemas que siguen los principios REST.

Ruby Lenguaje de script (no compilado), totalmente orientado a objetos, multiplataforma y de código abierto.

RubyGems Gestor de paquetes para el lenguaje de programación *Ruby* que proporciona un formato estándar y autocontenido (llamado gem) para poder distribuir programas o librerías en *Ruby*, una herramienta destinada a gestionar la instalación de éstos, y un servidor para su distribución.

Ruby on Rails Entorno de desarrollo web.

Screencast Grabación digital de la salida por pantalla de la computadora, a veces conteniendo narraciones de audio.

Scrum Marco de trabajo para la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software.

SGBD Agrupación de programas que sirven para definir, construir y manipular una base de datos.

Sprint Período en el cual se lleva a cabo el trabajo en sí.

SQL Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en estas.

TCP/IP Conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre computadoras.

Tex Sistema de tipografía.

Tienda virtual Sitio web donde se pueden comprar productos y pagarlos por cualquier medio electrónico.

UTF-8 Formato de codificación de caracteres Unicode e ISO 10646.

WBS Herramienta fundamental en la gestión de proyectos. Es una descomposición jerárquica del trabajo que va a ejecutar el equipo de proyecto, para cumplir con los objetivos de éste.

W3C Comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo.

WinEdt Editor de texto potente y versátil para Windows con una fuerte predisposición hacia la creación de documentos LaTeX.

Wireframe Boceto que define el contenido y la funcionalidad de una web.

World Wide Web *(WWW)* Sistema de distribución de información basado en hipertexto o hipermedios enlazados y accesibles a través de Internet.

XML Formato estándar para el intercambio de datos basado en archivos de texto plano con una estructura de etiquetas *(«tags»)*.

YAML Formato para guardar objetos de datos con estructura de árbol.

C. Gemas

A continuación describo las diferentes gemas que he utilizado para el desarrollo de la aplicación. El código de cada una de ellas se puede localizar en ***GitHub***.

- **Activemerchant [Luetke, Tobias; Fauser, Cody]:** Esta gema será utilizada para la facturación de los pedidos, ya que conecta con la pasarela de pago para gestionar las transacciones de los clientes a través de su tarjeta de crédito. (https://github.com/Shopify/active_merchant).
- **Acts-as-taggable-on [Kramarenko, Artem]:** Nos va a permitir añadir etiquetas a las obras para poder realizar las recomendaciones de obras similares a los clientes. (https://github.com/mbleigh/acts-as-taggable-on).
- **Authlogic [Johnson, Ben; ASCIIcasts]:** Esta gema sirve para gestionar la autenticación en la aplicación. Trabaja sólo con el código que gestiona la autenticación de los usuarios, ganando con ella flexibilidad y control acerca de cómo se presenta ésta al usuario. (https://github.com/binarylogic/authlogic).
- **Authorize-net [Authorize.net]:** Será utilizada para la facturación de los pedidos. A través de ella se gestiona la presentación de las transacciones de pago a las redes de procesamiento en nombre de los clientes. Esta basado en el Protocolo de Internet (IP), permitiendo a los comerciantes autorizar, abonar y gestionar las transacciones de cheques electrónicos y tarjetas de crédito desde una web. (http://rubydoc.info/gems/authorize-net/1.5.2/frames).
- **Country-select [Koziarski, Michael; Dean Shepherd, James]:** Nos proporciona una lista de países en la que se puede seleccionar uno. Esta lista está basada en la norma *ISO 3166*. La utilizaré para que el cliente pueda seleccionar su país en el momento de realizar la facturación de su pedido. (https://github.com/jamesds/country-select).
- **Globalize3 [Stachewicz, Tomasz]:** Provee servicios para la internacionalización y localización en *Rails*. Facilita el poder realizar traducciones sobre los modelos y vistas de la aplicación. Es compatible con la API I18n de *Rails*. (https://github.com/svenfuchs/globalize3).
- **Mail [Lindsaar, Mikel; ASCIIcasts]:** Es una biblioteca de Internet para *Ruby* diseñada para manejar de forma sencilla todas las funciones del correo electrónico. Las acciones de red se realizan a través de protocolos como: SMTP, POP3... Esta gema es utilizada por *Action Mailer* para crear los servicios de e-mail. (https://github.com/mikel/mail).
- **MetaSearch [Miller; Railscasts]:** Esta gema añade una forma muy útil de realizar búsquedas sobre un modelo desde un formulario. (https://github.com/ernie/meta_search).

- **Newrelic_rpm:** Es una sistema para el monitoreo y análisis del rendimiento de aplicaciones que es ofrecida como un servicio en la web. (https://github.com/newrelic/rpm).

- **Paperclip [Sichanugrist, Prem; Moses]:** Permite adjuntar archivos de cualquier tipo a un modelo de *ActiveRecord*, concretamente, en mi aplicación la he utilizado para adjuntar las fotos de los artistas al modelo *"artistum"* y las fotos de las obras de arte al modelo *"obra"*. Su implementación ha sido sencilla ya que los archivos son tratados como un atributo más del modelo.

  A continuación veremos, a modo de ejemplo, como le he indicado al modelo *"obra"* que deseo un archivo adjunto con el nombre de *"imagen"*.

```
class Admin::Obra < ActiveRecord::Base
  ....

  ### Paperclip - Almacenamiento en Dropbox ###
  has_attached_file :imagen,
        :storage => :Dropboxstorage,
        :path => "/:attachment/:id/:style.:extension",
        :url => "/:attachment/:id/:style.:extension",
        :styles => {
                      :thumb =>  "50x50#",
                      :medium => "100x100>",
                      :large =>  "300x300>"
                    }
  ....
end
```

Código 10.1: Implementación de la gema *"Paperclip"* en el modelo *"obra"*

  Esta gema no sólo se encarga de guardar el archivo en nuestro servidor, sino también crea automáticamente thumbnails[1] de esa imagen, las cuales me vienen muy bien para la aplicación. Para ello, *Paperclip* utilizar la librería *ImageMagick*[2] que es la realmente permite generar los thumbnails de las imágenes. (https://github.com/thoughtbot/paperclip).

- **Paperclipdropbox [Ketelle, Paul]:** Esta gema la he utilizado para poder tratar la gema *Paperclip* mediante el servicio de alojamiento de archivos *Dropbox*. (https://github.com/dripster82/paperclipdropbox).

[1] *Thumbnails:* Vistas en miniatura.
[2] *ImageMagick:* Librería para realizar manipulaciones de imágenes.

- **Will_paginate [Marohnic, Mislav]:** Proporciona un método que, a partir de un conjunto de elementos y un número de elementos por páginas, limita automáticamente el número de elementos mostrado por cada página y añade un paginador. (https://github.com/mislav/will_paginate/wiki).

D. Lenguaje Ruby

Ruby es un lenguaje de programación creado, en el año 1995, por el japonés *Yukihiro Matsumoto (Matz)*, el cual lo ha enfocado a la simplicidad y a la productividad, lo que favorece un desarrollo rápido y sencillo de aplicaciones. Presenta una sintaxis elegante y natural, haciéndolo muy fácil de leer y entender. Es un *lenguaje de script* (no compilado), totalmente *orientado a objetos*, multiplataforma y de código abierto (open source, software libre).

Ha combinado muchas de las características positivas de Perl, PHP, Java, C, Smalltak, Lisp haciendo un lenguaje bastante potente para el desarrollo de aplicaciones.

Es considerado un lenguaje muy intuitivo casi a un nivel de lenguaje humano, asemejándose al lenguaje natural, de esta forma hace que la experiencia de programación sea más divertida.

La filosofía de *Ruby* es `Don't repeat yourself (DRY)` *(«No te repitas»)*, es decir, la idea de *Ruby* es que no necesitamos repetir lo que ya ha definido en otro lugar. Esto hace a *Ruby* muy compacto.

A continuación veremos algunas de las *características* más importantes de Ruby.

- **Orientado a Objetos**: *Ruby* es un lenguaje totalmente orientado a objetos, es decir, que sigue estrictamente el *paradigma de la Programación Orientada a Objetos (POO)*.
- **Herramientas de creación de tareas y gestión de dependencias**: Ruby dispone de una herramienta llamada *Rake*[3] que permite la creación de tareas repetitivas y la gestión de dependencias de las tareas de mantenimiento.
- **Multiplataforma**: *Ruby* dispone de dos implementaciones de intérpretes:
  - el oficial de Ruby, que es el más utilizado,
  - y el JRuby, el cual es una implementación realizada en Java que funciona en una máquina virtual Java.

  Ambas implementaciones del intérprete las podemos encontrar en multitud de sistemas operativos como: Linux, Mac OS X, Microsoft Windows, Windows CE y la mayoría de Unix. Además desde la versión 1.9 Ruby ha sido portado a Symbian OS 9.x.

[3] *Rake* es el equivalente a *make* de GNU para *Ruby*.

- **Entornos de desarrollo Web *(Frameworks)***: *Ruby* dispone de diversos frameworks para el desarrollo de aplicaciones Web. El más famoso y más completo es **Rails** que junto con **Ruby** forman la plataforma de desarrollo ***Ruby on Rails***, la cual ha sido la escogida para realizar este proyecto.

- **Software Libre**: *Ruby* dispone de una *licencia GPL*, creada por la *Free Software Foundation* la cual hace que este lenguaje de programación sea software libre y por tanto, los usuarios tienen total libertad para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.

- **Gestión automática de memoria**: *Ruby* dispone de un "recolector de basura" que gestiona automáticamente la reserva de memoria del ordenador. Gracias a la gestión automática de memoria no se ha de invocar ninguna subrutina para liberar espacios de memoria ocupados, ya que es el propio intérprete de *Ruby* el que gestiona el ciclo de vida de los objetos creados.

- **Generador de documentación**: *Ruby* dispone de un generador de código por defecto llamado **Rdoc**. Este parsea los archivos *.rb, .rbw* y *.c* del proyecto y busca los comentarios situados encima de las definiciones de las funciones. Con estos datos se crean archivos *HTML*, formando una fuente de documentación de librerías muy útil a la hora de reutilizar código.

- **Lenguaje Interpretado**: Al ser un lenguaje totalmente interpretado, no requiere ser compilado antes de ejecutarse, ya que la ejecución del código se realiza a través de un intérprete que crea una representación intermedia del código que se va compilando a medida que va a ser ejecutado.

- **Dinamismo**: En *Ruby*, las clases nunca están cerrada, esto quiere decir, que siempre se pueden añadir nuevos métodos a esta clase. Siendo válido tanto para las clases incluidas con el intérprete como para las nuevas clases escritas.

- **Tipado Dinámico**: *Ruby* es un lenguaje de tipos débiles, es decir, que las variables que usa no tienen un tipo fijo y no es necesaria su declaración para poder usarlas. En *Ruby*, se confía menos en el tipo de un objeto y más en sus capacidades, que significa que el tipo de un objeto está definido por lo que puede hacer, no por lo que es.

  El *tipado dinámico* aporta flexibilidad a la hora de programar, ya que libera al programador de la obligación de tener que declarar todas las variables usadas en el desarrollo del código. Esto conlleva la creación de programa con menor número de líneas y en consecuencia un aumento de productividad. Pero se puede convertir en una desventaja, ya que el uso del tipado dinámico puede comportar la generación de pequeños bugs difícilmente detectables hasta la ejecución del programa. Por esta razón, el desarrollo de aplicaciones en *Ruby* requieren que se creen multitud de test para poder detectar el correcto funcionamiento del código generado.

- **Sintaxis concisa, compacta**: La Sintaxis de *Ruby* permite escribir bloques de código muy compactos y en un lenguaje muy "humano". En consecuencia permite crear programas con menor número de líneas de código que se refleja en una menor cantidad de puntos de posibles errores de tipado.

E. Metodología ágil

Una *Metodología Ágil* es una metodología efectiva para modelar y documentar un proyecto de software. La forman una colección de *valores, principios* y *prácticas* para modelar software, que pueden ser aplicados de manera simple y ligera.

Esta metodología tiene varios *principios* que la diferencian de las metodologías tradicionales, los cuales están reflejados en el ***Manifiesto Ágil*** [Beck, Kent; Beedle, Mike; Bennekum, Arie; Otros] . Según éste se valora:

- ***Al individuo y las interacciones*** del equipo de desarrollo sobre el *proceso y las herramientas*. Las personas son el principal factor de éxito de un proyecto software. Es mejor crear un buen equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- ***Desarrollar software que funciona*** más que conseguir una *buena documentación*. La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental.
- La ***colaboración con el cliente*** más que la *negociación de un contrato*. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- ***Responder a los cambios*** más que *seguir estrictamente un plan*. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc. . .) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación debe ser flexible y abierta.

Los valores anteriores inspiran los doce ***Principios del Manifiesto*** [Beck, Kent; Beedle, Mike; Bennekum, Arie; Otros]. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo.

Metodología ágil SCRUM

Scrum fue desarrollado por *Ken Schwaber, Jeff Sutherland* y *Mike Beedle* y es un marco de referencia para generar una metodología ágil. Su principal objetivo es llevar los proyectos a su realización final cuando están envueltos en entornos cambiantes, como es el mundo del software y, especialmente, el entorno web.

Scrum es un proceso en el que se aplican de manera regular un conjunto de *buenas prácticas* para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en el estudio de la manera de trabajar de equipos altamente productivos.

En *Scrum* se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, está especialmente indicado para proyectos en entornos complejos, donde:

- se necesita obtener resultados pronto,
- los requisitos son cambiantes o poco definidos,
- la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

También se utiliza para resolver situaciones cuando:

- no se está entregando al cliente lo que necesita,
- las entregas se alargan demasiado,
- los costes se disparan o la calidad no es aceptable,
- se necesita capacidad de reacción ante la competencia,
- la moral de los equipos es baja y la rotación alta,
- es necesario identificar y solucionar ineficiencias sistemáticamente o
- se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

La metodología de *Scrum* se sustenta sobre tres principios de igual importancia: *transparencia, inspección* y *adaptación*.

- **Transparencia** hace referencia a que todas las personas implicadas deben estar al corriente de los cambios, contratiempos o impedimentos que surjan.
- La **inspección** quedará cubierta por medio de las reuniones y trabajos en equipo que plantea la metodología, para detectar las desviaciones con la mayor brevedad posible.
- **Adaptación** es uno de los principios básicos de la metodología ágil y uno de los objetivos por los que se aplica *Scrum*. Consiste en introducir los cambios que puedan surgir con la menor implicación a los objetivos del proyecto.

F. Pruebas en Rails

La comprobación del buen funcionamiento de mi aplicación en *Rails* la he desarrollado mediante las pruebas que a continuación describo:

- **Pruebas unitarias**: Son una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. En *Rails* estas pruebas se aplican a relaciones entre modelos, así como a sus validaciones.
- **Pruebas funcionales**: Son unas pruebas basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. En *Rails* verifican los efectos de la llamada directa a un método de un controlador.
- **Pruebas de integración**: Son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. Es la fase en la cual módulos individuales de software son combinados y probados como un grupo. Sirven para probar interacciones entre los elementos. En *RoR* se prueba la interacción entre los controladores.

G. Rails

Rails es un framework de aplicaciones web de código abierto, multiplataforma y distribuido bajo la licencia del MIT. En otras palabras, es una serie de utilidades y de herramientas para crear aplicaciones web más rápidamente que haciéndolo desde cero.

Fue desarrollado por el danés *David Heinemeier Hansson*, como una herramienta para facilitarse el trabajo al programar la aplicación web *Basecamp* para la empresa *37 Signals*. En julio de 2004 liberó el código como software libre, y en febrero de 2005 comenzó a aceptar colaboraciones para mejorar *Rails*, formándose un amplio equipo de programadores, el *Core Team*.

Rails utiliza el lenguaje **Ruby** siguiendo el paradigma de la arquitectura *Modelo-Vista-Controlador (MVC)* el cual explicaré más adelante (ver párrafo *4.1.2 "Arquitectura MVC en Rails"* de la *página 104*).

Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

El lenguaje de programación *Ruby* permite la metaprogramación, de la cual *Rails* hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible.

Rails se distribuye a través de *RubyGems*, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones *Ruby*.

Características de Rails

Algunas de las características de *RoR* son:

- Arquitectura *MVC*.
- Aprovecha la Metaprogramación de *Ruby*.
- Simplicidad.
- Posee un potente motor de generación de código.
- Conexión a varios motores de BD.

- Manejo de cambios a BD a través de migraciones.
- No usa directamente *SQL* en las consultas a BBDD (pero se puede).
- *AJAX* integrado (*jQuery*).
- Gran cantidad de Helpers (ayudantes) para generar elementos repetitivos.
- Posee varias tareas *Rake* predefinidas.
- Maneja el ruteo de manera fácil y dinámico.
- Soporte integrado a Internacionalización *(i18n)* y Localización.

Soportes de Rails

- **Plataformas**: GNU/Linux, Unix, FreeBSD, Mac OS X, Windows.
- **Bases de Datos**: PostGreSQL, MySQL, Oracle, SQL Server, SQLite, IBM DB2.
- **Servidores Web**: Webrick (Servidor integrado con Rails), Apache, Lighttpd.

Filosofía

La filosofía de *Ruby on Rails*, se basa en dos principios fundamentales de programación:

- ***Don't Repeat Yourself***: Las definiciones solo deben de realizarse una única vez, pues los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Cada pieza de conocimiento en un sistema, deberá ser expresada en un sólo lugar.

  Este principio se basa en escribir menos líneas de código para implementar la aplicación. Si el código es pequeño quiere decir que el desarrollo es más rápido y con menos errores, lo que hará que el código sea fácil de entender, mantener y mejorar.

  Un *ejemplo* de esto es el utilizado por el *patrón* ***ActiveRecord***, *Ruby* no necesita que se le especifiquen los campos (columnas) de un objeto, este es capaz de obtenerlos a partir de la base de datos, de modo que no es necesario replicar dicha información en el código.

- ***Convention Over Configuration***: Gracias a este principio el usuario solo necesita especificar aquellas clases que tienen un comportamiento diferente. Por *ejemplo*, si en el modelo existe una *clase Persona* esta tendrá su correspondiente *tabla personas*, de modo que si queremos usar una clase con dicho comportamiento no debemos especificar nada nuevo, a menos que queramos crear un nuevo *tipo Médico* que tenga aspectos que difieran de una *Persona*.

Llevando estás dos sencillas reglas a la práctica, será muy sencillo realizar aplicaciones web funcionales, con una sencillez y claridad en el código que no se puede hacer en otros lenguajes.

Estructura de una aplicación Rails

En la siguiente imagen podemos apreciar cómo se gestiona *Rails* internamente:

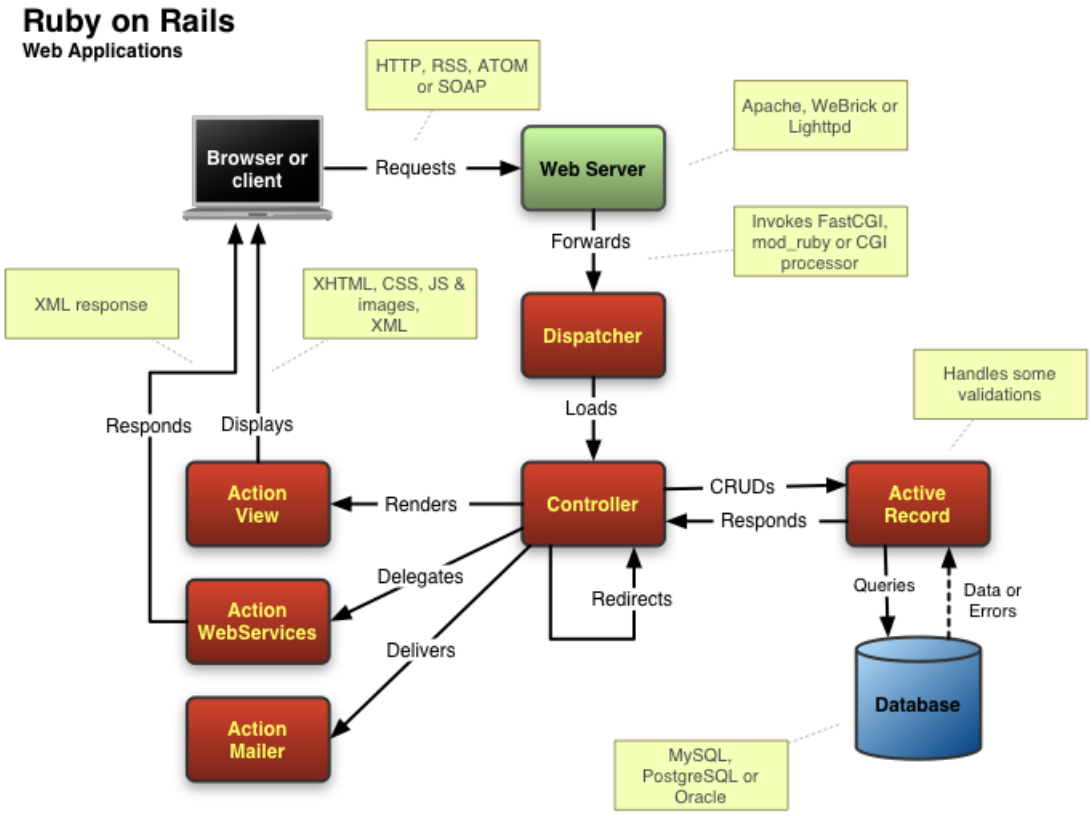


Figura 10.1: Diagrama de flujo de Rails

A continuación voy a describir algunos de los módulos que aparecen en la imagen:

- **Active Record**: Es el módulo que se encarga de la conexión entre la aplicación y la base de datos. La base de los modelos de datos. Proporciona independencia de la BD, relaciona modelos, funcionalidad básica *CRUD*, capacidad de búsqueda avanzada, ...
- **Action View**: Este módulo se encarga de renderizar los datos que el controlador quiere mostrar al usuario. Su uso es mediante vistas que pertenecen a acciones del controlador. Además da la opción de hacer subvistas para la eficiencia de nuestra aplicación. Otra cosa a tener en cuenta es la opción de poner layouts, que son vistas

que son visibles para cualquier acción del controlador, o si lo ponemos en el principal será vista durante toda la web, salvo que no se quiera. Puede crear *HTML* y *XML*.

- **Action Controller**: Es la base de la aplicación. Maneja los controladores de la aplicación. Procesa peticiones, extrae parámetros y ejecuta.
- **Action Mailer**: Es el módulo encargado de crear servicios de email. Se puede usar para enviar, recibir y procesar éstos.

  Su interfaz es sencilla gracias a una rápida configuración del servidor (que veremos más adelante). Uno de sus puntos fuertes es poder crear vistas para las acciones de email a mandar, se trabaja como si fueran vistas web que el sistema se encarga de enviar a su destinatario.
- **Action WebServices**: Es el módulo que se encarga de las operaciones de WebServices de la aplicación, si se desean utilizar. Lo que consigue es que se pueda llamar a las acciones definidas en los controladores mediante llamadas al sistema y no a través de las vistas. Tiene sentido, el uso de este módulo, cuando se quiere que la aplicación web puede ser accedida de forma directa por otro programa vía internet.
- **WebServer**: Servidores para el desarrollo y las pruebas, en mi caso utilizaré WEBrick.

Herramientas y componentes útiles

Rails proporciona una serie de herramientas que nos facilitan las tareas comunes. Estas son:

- **Scaffolding**: Método de meta-programación que genera el *modelo, vista* y *controlador* correspondiente a un recurso de la aplicación en una sola operación y de forma automática. Trataré esta herramienta un poco más en la *Sección 5.2 "Esqueleto para la aplicación"* de la *página 139*.
- **Rake** [Fowler, Martin]: Herramienta de generación o automatización de código escrito en *Ruby*. Los *"Rakefiles*[4]" utilizan la sintaxis del lenguaje *Ruby* con lo que no es necesario aprender ninguna sintaxis específica de herramientas de generación de código.
- **WEBrick**: Librería de *Ruby* que provee de un servidor *HTTP* sencillo, el cual es utilizado para probar aplicaciones en un entorno de desarrollo.

[4]Los *Rakefiles* equivalen a los *Makefiles* de *Make*.

Para la creación de aplicaciones web *Rails* posee los siguientes *componentes*, la mayoría de ellos descritos en la sección anterior:

- **Active Record**
- **Action View**
- **Action Controller**
- **Action Mailer**
- **Action Resource**: Es un framework que gestiona la conexión entre objetos de negocio y RESTful [De Seta, Leonardo] web services. Implementa, con la semántica *CRUD*, el mapeo entre estos.
- **Railties**: Código del núcleo de *Rails* que crea nuevas aplicaciones y las conecta con los framework en una sola aplicación.
- **Active Support**: Gran colección de clases y extensiones de la biblioteca estándar de *Ruby*.

Base de datos en Rails

Rails:

- Soporta diferentes SGBDRs para almacenar los datos, incluyendo MySQL, PostgreSQL, SQLite, IBM DB2 y Oracle. Por defecto tiene la biblioteca SQLite.
- Gestiona los accesos a la base de datos automáticamente, aunque si se necesita se pueden hacer consultas directas en *SQL*.
- Intenta mantener:
  - la neutralidad con respecto a la base de datos,
  - la portatibilidad de la aplicación a diferentes sistemas de base de datos y
  - la reutilización de bases de datos preexistentes.

  Sin embargo, debido a la diferente naturaleza y prestaciones de los SGBDRs el framework no puede garantizar la compatibilidad completa.

H. Relación de Casos de Uso y Mockups

A continuación muestro un listado que indica para cada *Caso de Uso* cual es el mockup que lo implementa. Éstos están contenidos en la carpeta **mockups** del CD adjunto.

- *CU de Gestión de Artistas*

  Los CU de Gestión de Colecciones, Secciones y Técnicas son implementados mediante mockup similares a los de la Gestión del Artista.

  - listado_artistas
  - admin_edicion_artista
  - admin_ficha_artista
  - ficha_artista

- *CU de Gestión de Obras*

  - listado_obras
  - ficha_obra
  - foto_obra

- *CU de Gestión de Catálogo*

  - catalogo
  - busqueda
  - obras_recientes

- *CU de Gestión de Cesta de la Compra*

  - cesta_compra

- *CU de Gestión de Etiquetas*

  - listado_obras_por_etiquetas

- *CU de Gestión de Seguridad*

  - identificacion
  - listado_usuarios
  - registro_datos_identificacion
  - registro_datos_personales
  - registro_datos_confirmación
  - notificacion_registro_cliente
  - restablecer_contraseña_cliente

- *CU de Gestión de Facturación*
  - listado_pedidos
  - informacion_pedido
  - admin_mostrar_pedido
  - pago_tarjeta
  - transferencia_bancaria
  - confirmacion_pedido_cliente
  - solicitud_pedido_galeria
- *CU de Gestión de Internacionalización*
  - admin_traduccion_artista
  - admin_traducir_artista
- *CU de Quienes somos*
  - quienes_somos
- *CU de Condiciones*
  - condiciones
- *Otros*
  - inicio
  - panel_administracion
  - contactar

I. Test de Usabilidad

Tarea	Dificultad Fácil	Dificultad Media	Dificultad Alta	Tarea completada (si/no)	Tiempo empleado (minutos)	Nivel de satisfacción (0-10)
Registro de usuario						
Consultar catálogo						
Consultar características de una obra						
Buscar una obra concreta						
Insertar obras en la cesta						
Realizar una compra						
Crear una nueva obra de arte						

Figura 10.2: Test de usabilidad

Capítulo 11

Glosario de términos

Bibliografía

AENOR (2002).
Norma española: UNE 157001. Criterios generales para la elaboración de proyectos.
http://www.ehu.es/Degypi/General/norma157001.pdf.

Alarcos, UCLM-ESI. Grupo.
Gestión de riesgos en proyectos software.
http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/7/pgsi-t7.pdf.

ASCIIcasts.
160: Authlogic.
http://es.asciicasts.com/episodes/160-authlogic.

ASCIIcasts.
206: Action Mailer en Rails 3.
http://es.asciicasts.com/episodes/206-action-mailer-en-rails-3.

ASCIIcasts.
228: Sortable Table Columns.
http://asciicasts.com/episodes/228-sortable-table-columns.

ASCIIcasts.
240: Search, Sort, Paginate with AJAX.
http://asciicasts.com/episodes/240-search-sort-paginate-with-ajax.

Ataz López, Joaquín (2006).
Guía casi completa de BIBTEX.
ftp://ftp.ctan.org/tex-archive/info/spanish/guia-bibtex/guia-bibtex.pdf.

Authorize.net.
Gema authorize-net.
http://rubydoc.info/gems/authorize-net/1.5.2/frames/.

AuthorizeNet (2011).
Merchant Web Services API. Automated Recurring Billing^{TM} (ARB) XML Guide.
http://www.authorize.net/support/ARB_guide.pdf.

AVOS.
Delicious.
http://delicious.com/.

Beck, Kent; Beedle, Mike; Bennekum, Arie; Otros (2001a).
Manifesto for Agile Software Development.
http://agilemanifesto.org/.

Beck, Kent; Beedle, Mike; Bennekum, Arie; Otros (2001b).
Principles behind the Agile Manifesto.
http://www.agilemanifesto.org/principles.html.

Britt, James.
Module::Authlogic::I18n.
http://rubydoc.info/github/binarylogic/authlogic/master/Authlogic/I18n.

Brown, Mat.
Gema sunspot_rails.
https://github.com/outoftime/sunspot/tree/master/sunspot_rails.

BuenasTareas.com.
Patron Active Record.
http://www.buenastareas.com/ensayos/Patron-Active-Record/3226566.html.

Burgos Pintos, Aris; Cornejo Barrios, Alicia; Gámez Mellado, Antonio; Otros (2010).
Taller de LaTeX.

Center, AuthorizeNet. Developer.
AuthorizeNet.
https://github.com/binarylogic/authlogic.

Corporation, Oracle.
MySql.
http://dev.mysql.com/.

CraigsList (2012).
Craigslist Classifieds.
http://www.craigslist.org/.

De Seta, Leonardo (2008).
Introducción a los servicios web RESTful.
http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html.

Dodero, Juan Manuel.
DSLs en Ruby.
http://deki.uca.es/II_ApuntesRuby_on_Rails/Ruby_DSLs.

Fauser, Cody (2008).
PayPal Express Payments with ActiveMerchant.
http://www.codyfauser.com/2008/1/17/paypal-express-payments-with-activemerchant.

Fowler, Martin (2006).
Rake: algunos comandos útiles.
http://dummyonrails.lacoctelera.net/post/2007/06/13/rake-algunos-comandos-utiles.

Franco Aquino, Rafael (2011).
Una introducción a Ruby on Rails.
http://www.slideshare.net/movihus/rails-etyc-2011.

Fuchs, Sven.
Rails i18n.
https://github.com/svenfuchs/rails-i18n/tree/master/rails/locale.

Geeknet, Inc..
Documentation from Ruby Source Files.
http://rdoc.sourceforge.net/.

Geeknet, Inc..
freshmeat.net.
http://freshmeat.net/.

Global, Comercio Electrónico (2011).
Magento o Prestashop: ¿Cuál es el mejor software libre para tienda de comercio electrónico?.
http://e-global.es/software-libre-de-comercio-electronico/magento-o-prestashop-icual-es-el-mejor-software-libre-para-tienda-de-comercio-electronico.html.

Gómez López, Rubén (2010).
PHP vs Java.
http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=PHPVsJava.

Google.
Google Code.
http://code.google.com/.

Hartl, Michael (2010).
Ruby on Rails Tutorial. Learn Rails by Example.

Heinemeier Hansson, David.
Ruby on Rails.
http://rubyonrails.org/.

Hellsten, Christian; Laine, Jarkko (2006).
Beginning Ruby on Rails E-Commerce. Apress.

Heroku.
Getting Started with Heroku.
http://devcenter.heroku.com/articles/quickstart.

IEEE (1990).
IEEE Standard Glossary of Software Engineering Terminology.
http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf.

IEEE-SA (1998).
IEEE 830/1998. IEEE Recommended Practice for Software Requeriments Specifications.
http://www.slideshare.net/ahmedhasan/ieee-830-1998-software-requirements-specifications.

Indeed (2012).
Indeed.
http://www.indeed.com/.

Johnson, Ben.
Gema authlogic.
http://rubydoc.info/gems/authlogic/3.1.0/frames.

Johnson, Ben (2012).
Authlogic.
https://github.com/binarylogic/authlogic.

Ketelle, Paul.
Gema paperclipdropbox.
https://github.com/dripster82/paperclipdropbox.

Koziarski, Michael; Dean Shepherd, James.
Gema country-select.
https://github.com/jamesds/country-select.

Kramarenko, Artem.
Gema acts-as-taggable-on.
http://rubydoc.info/gems/acts-as-taggable-on/2.2.2/frames.

Leonardo Lemus, Jorge; Navas Muños, Jenniffer (2009).
Manual OpenProj.
http://www.uco.es/ lr1maalm/Manual-openproj.pdf.

Lindsaar, Mikel.
Gema mail.
https://github.com/mikel/mail.

Luetke, Tobias.
Gema activemerchan.
http://activemerchant.org/.

MacAulay, James; Lutke, Tobi; Fauser, Cody; Baker, Jimmy.
Gema active_shipping.
https://github.com/Shopify/active_shipping.

Marcelo Hernán, Schenone (2004).
Tesis: Diseño de una Metodología Ágil de Desarrollo de Software.
http://materias.fi.uba.ar/7500/schenone-tesisdegradoingenieriainformatica.pdf.

Marohnic, Mislav.
Gema will_paginate.
https://github.com/mislav/will_paginate/wiki.

Matake, Nov.
Gema paypal-express.
https://github.com/nov/paypal-express.

Matsuda, Akira.
Gema kaminari.
https://github.com/amatsuda/kaminari.

Microsistems, Sun.
MySQL.
http://www.mysql.com/.

Miller, Ernie. *Gema meta_search.*
https://github.com/ernie/meta_search.

Millán, Emmanuel N.; McCreesh, John (2005).
Cuatro días con Rails..

Moses, Brooks (2007).
The listings package.
ftp://ftp.tex.ac.uk/tex-archive/macros/latex/contrib/listings/listings.pdf.

Motion, Sparkle.
Gema nokogiri.
http://nokogiri.org/.

PAE. Portal Administración Electrónica (2011).
Métrica v.3.
http://administracionelectronica.gob.es/archivos/pae_000001040.pdf.

Paypal.
Incorporar pago de PayPal a su carrito de compras de terceros.
https://www.paypal.com/.

PayPal (2008).
Website Payments Standard Integration Guide.
https://www.paypalobjects.com/en_US/pdf/PP_WebsitePaymentsStandard_IntegrationGuide.pdf.

Quaranto, Nick; Otros.
RubyGems.org.
http://rubygems.org/.

Railscasts.
251 MetaWhere and MetaSearch.
http://railscasts.com/episodes/251-metawhere-metasearch?language=es&view=asciicast.

Ross, Phil.
Gema tzinfo.
http://tzinfo.rubyforge.org/.

Ruby, Sam; Thomas, Dave; Heinemeier Hansson, David (2010).
Agile Web Development with Rails. The Pragmatic Bookshelf, 4ª edición.

Sichanugrist, Prem.
Gema paperclip.
https://github.com/thoughtbot/paperclip.

Simonic, Aleksander.
WinEdt.
http://www.winedt.com/.

Software, Black Duck.
Ohloh.
http://www.ohloh.net/.

Somoza, Gabriel (2009).
Vía Rails. Cómo Adjuntar Archivos a un Modelo en Rails. Paperclip.
http://viarails.wordpress.com/2009/11/29/como-adjuntar-archivos-a-un-modelo-en-rails/.

Stachewicz, Tomasz.
Gema globalize3.
https://github.com/svenfuchs/globalize3.

StuFF mc.
Demo globalize2.
https://github.com/joshmh/globalize2-demo.

Thomas, Dave. *Api Ruby on Rails.*
http://api.rubyonrails.org/.

Venkatreddy Dwarampudi; Shahbaz Singh Dhillon; Jivitesh Shah; Nikhil Joseph Sebastian; Nitin Satyanarayan Kanigicharla (2010).
Comparative study of the Pros and Cons of Programming languages.
http://arxiv.org/pdf/1008.3431.pdf.

W3C.
World Wide Web Consortium (W3C).
http://www.w3.org/.

Welton, David N. (2005).
The Economics of Programming Languages.
http://www.welton.it/articles/programming_language_economics.html.

Welton, David N. (2011).
Programming Language Popularity.
http://www.langpop.com/.

Wikibooks (2011).
LATEX Wikibook.
http://en.wikibooks.org/wiki/LaTeX.

Wikipedia.
Desarrollo ágil de software.
http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software.

Wikipedia.
Lenguaje de programación interpretado.
http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_interpretado.

Wikipedia.
Pruebas de software.
http://es.wikipedia.org/wiki/Pruebas_de_software.

Wikipedia.
Ruby.
http://es.wikipedia.org/wiki/Ruby.

Wikipedia.
Ruby on Rails.
http://es.wikipedia.org/wiki/Ruby_on_Rails.

Yahoo.
Yahoo Search.
http://es.search.yahoo.com/.

GNU Free Documentation License

Version 1.3, 3 November 2008

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to

any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called .ºpaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeXinput format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To "Preserve the Title.of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying

with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "Historyïn the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History"section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- For any section Entitled .Acknowledgements.or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section Entitled .Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section to be Entitled .Endorsements.or to conflict in title with any Invariant Section.
- Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled .[En]dorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License .ºr any later version.ªpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Sit") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace "the with ... Texts". line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.